VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Framework editace 2D grafiky pro iOS

2D Graphics Editing Framework for iOS

2013 **Radim Halfar**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Radim Halfar**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Framework editace 2D grafiky pro iOS 2D Graphics Editing Framework for iOS

Zásady pro vypracování:

Student zmapuje současný trh frameworků pro zařízení iOS vhodných pro tvorbu editace 2D grafiky. Student se zaměří na užití zvoleného frameworku a vytvoří vlastní rozšíření frameworku pro tvorbu grafických aplikací, buď v prostředí MacOS X (Objective C nebo C++), eventuálně Windows (C++).

1. Student prozkoumá současné trendy na poli vývoje editorů grafických aplikací pro iOS.
2. Student navrhne a implementuje technologii pro účely editace 2D grafiky a jejího publikování na webu a sociálních sítích.
3. Student navrhne a vytvoří jednoduché API pro vývoj iOS aplikací vhodných pro editaci grafiky a její sdílení na webu a sociálních sítích.
4. Student vytvoří na základě požadavků z ad 2) a API z ad 3) ukázkové aplikace vhodné pro další zobecnění pro budoucí softwarová řešení pro iOS mobilní zařízení.

Seznam doporučené odborné literatury:

[1] KOCHAN, Stephen G. Programming in Objective-C 2.0. 2nd ed. Upper Saddle River: Addison Wesley Professional, c2009, xv, 600 s. ISBN 978-0-321-56615-7.
[2] MARK, Dave. Beginning iPhone development: exploring the iPhone SDK. Berkely: Apress, c2009, xxiii, 508 s. ISBN 1430216263.
[3] STROUGO, Rod a Ray WENDERLICH. Learning Cocos2D: a hands-on guide to building iOS games with Cocos2D, Box2D, and Chipmunk. Upper Saddle River, NJ: Addison-Wesley, c2012, xxxv, 590 p. ISBN 03-217-3562-5.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: *30.04.2013*

……..……

podpis studenta

Poděkování

Rád bych poděkoval Ing. Davidu Ježkovi, Ph.D. za odbornou pomoc, konzultaci a mnoho přínosných poznatků při vytváření této bakalářské práce, mé matce, ze neustálou podporu a pochopení všech aktivit při mém studiu a v neposlední řadě společnosti AstrumQ Interactive, s.r.o. za cenné poznatky spojené s vývojem mobilních aplikací a také za poskytnutí potřebných nástrojů pro vývoj a testování.

Abstrakt

V současné době se vývoj mobilních aplikací stává čím dál tím více populárnějším. Jednou z nejoblíbenějších platforem je iOS, jenž vývojářům umožňuje rozsáhlé možnosti realizace požadovaných řešení. V této práci se zaměříme na grafické možnosti systému iOS, a to konkrétně na možnosti tvorby a editace 2D grafiky na zařízeních iPhone a iPad. Srovnáme již dostupné grafické frameworky pro tvorbu a editaci 2D grafiky a vytvoříme vlastní řešení této problematiky. Budeme se snažit pokrýt požadované vlastnosti pro knihovny tohoto typu. Uvedeme také stručný přehled možných přístupů pro řešení problematiky. Své vlastní řešení demonstrujeme na ukázkové aplikaci.

Klíčová slova

iOS, Core Graphics, Cocoa Touch, OpenGL, CGContext, malování, plnící algoritmy, Objective – C, iPhone, iPad, 2D grafika

Abstract

Lately, the development of mobile applications became very popular. One of the most popular platform for the development is iOS, which allows wide opportunities of raelization the desired solutions to developers. In this thesis, we will focus on graphic capabilities of the iOS platform, specifically on the opportunities of editing and creating 2D graphics on the devices iPhone and iPad. We will compare existing frameworks which support desired funcionality and we will create our own solution. We will try to cover desired capabilities for the libraries focusing on this problematics. We will state overview of the accesses for solving this problem. Our solution will be used in the sample application.

Key words

iOS, Core Graphics, Cocoa Touch, OpenGL, CGContext, painting, filling algorithms, Objective – C, iPhone, iPad, 2D graphics

Seznam použitých zkratek a symbolů

Zkratka	Anglický význam	Český význam
API	Application Programming Interface	Rozhraní programovacího jazyk
Apple AppStore	Proprietary service of Apple Inc.	Proprietární služba společnosti Apple
ARC	Automated Reference Counting	Automatická správa paměti
Buffer	Memory for temporary files	Vyrovnávací paměť
BSD licence	Licence for free software	Svodobná licence určená pro software
CA	Core Animation	Framework společnosti Apple
CG	Core Graphics	Grafický framewrok společnosti Apple
CMD	Keyboard shotcut equivalent to ctrl	Klávesa ekvivalentní ctrl(control)
Context	Graphic structure used for rendering	Grafická struktura určená pro vykreslování
Framework	Software structure used as support within programming	Softwarová struktura využívaná jako podpora při programování
GC	Garbagge collector	Sběratel volné paměti
GL	Graphic Library	Grafická knihovna
GPU	Graphic processing unit	Grafický procesor
IDE	Integrated Development Enviroment	Vývojové prostředí
iOS	Operating system used in devices of Apple Inc., iPhone, iPad, iPod	Operační systém společnosti Apple využívaný v zařízeních iPhone, iPad a iPod
JSON	JavaScript Object Notation	Java skriptový objektový zápis
SDK	Software Development Kit	Soubor funkcí a knihoven zajišťující určitou funkcionalitu
UI	User interface	Uživatelské rozhraní
XCode	IDE for creating applications	IDE pro tvorbu aplikací

Obsah

Seznam použitých obrázků

Výpisy zdrojových kódů

Tabulky

1 Úvod

Cílem této práce je analyzovat vybrané frameworky a technologie pro tvorbu a editaci 2D grafiky pro platformu iOS a vytvořit vlastní řešení této problematiky. V následujícím textu bude slovem aplikace rozuměna iOS aplikace plně kompatibilní s zařízeními iPhone, iPad a iPod Touch. Pokud nebude uvedeno jinak, všechny technologie a frameworky uvedené v textu budou plně kompatibilní s zařízeními iPhone, iPad a iPod Touch.

V první kapitole budou uvedeny zdroje, ať hardwarové či softwarové, potřebné pro vývoj aplikací.

V následných kapitolách bude od čtenáře očekávána základní znalost iOS SDK a mírně pokročilá znalost programovacího jazyka Objective – C.

Druhá kapitola bude věnována současně dostupným technologiím pro editaci 2D grafiky. V této části budou vybrané technologie detailněji popsány a srovnány. Dále budou analyzovány vybrané frameworky, které využívají dostupné technologie.

Třetí kapitola bude věnována návrhu řešení problematiky tvorby a editace 2D grafiky. V této kapitole bude odůvodněn výběr technologie a frameworku. Dále zde bude provedena analýza požadavků pro vytvoření vlastního, či rozšíření již existujícího frameworku. Následující část textu bude věnována objasnění funkcionality zvolené technologie a frameworku.

Čtvrtá kapitola bude věnována implementaci návrhu popsaného ve čtvrté kapitole. V závěru této kapitoly dojde ke shrnutí a popsání implementovaného API. Společně s popisem, budou uvedeny a odůvodněny optimalizace provedené během vývoje.

V závěru shrneme úspěchy této bakalářské práce, uvedeme skutečnosti, jenž tato bakalářská práce obsahuje a popíšeme možná rozšíření vytvořené aplikace.

2 Vývoj mobilních aplikací pro platformu iOS

V dnešní době hektického světa plného chytrých zařízení se trendy v programování ubírají směrem, jenž dávno přesahuje hranice představivosti z dob konce 20. století. Na trhu aplikací existuje několik různých platforem. Z mého pohledu je nejzajímavější platformou iOS společnosti Apple. V České republice je vývoj pro tuto platformu ještě stále v začátcích, a proto jsem se také rozhodl pro zrealizování této bakalářské práce.

2.1 Vyjímečnost iOS

Systém iOS je zcela uzavřeným systémem společnosti Apple. Jeho unikátnost spočívá v tom, že je zcela zaměřen na veškeré detaily, jenž může koncový uživatel ocenit. Avšak největší devízou tohoto systému je, jak uvádí sama společnost Apple, stabilita jako skála[21] . Současné verze systému je iOS 6, konkrétně tedy iOS 6.1.3[1]. Důkazem rostoucího zájmu o vývoj iOS aplikací je jejich počet v AppStore. Oficiální data uvádějí, že počet aplikací pro iPhone a iPad v lednu roku 2013 činil 775 000 z čehož 335 000 je oficiálně uváděno jako nativních pro iPad.

Nespornou výhodou vývoje pro tuto platformu je přesně definované množství zařízení. V případě této platformy uvažujeme zařízení iPod Touch, iPhone a iPad.[22]

Obrázek 1: Apple iPad 4-té generace[2]

[1] Zdroj http://support.apple.com/kb/ht1222

[2] Zdroj http://www.apple.com/cz/ipad/specs/

Obrázek 2: Apple iPhone 5[3]

Obrázek 3: Apple iPod Touch[4]

Díky této nesporné výhodě, se vývojáři mohou plně soustředit na maximální výkon jejich aplikací a nemusí se složitě zabývat designováním UI pro své aplikace, jako je tomu třeba u konkurenční platformy Android. Zařízení iOS se staly velmi lákavými pro koncové uživatele nejen díky své funkčnosti, ale také díky nadčasovému designu.

2.2 Vývoj aplikací

Vývoj aplikací pro iOS není určen pro každého. Programátoři musí nevyhnutelně splňovat podmínky, jenž jsou stanoveny společností Apple.

2.2.1 Hardware a software požadavky

Nevyhnutelnou podmínkou pro vývoj iOS, potažmo Mac OS X aplikací, je hardware, který podporuje software společnosti Apple. Podporovaným softwarem pro vývoj rozumíme všechny nástroje poskytované společností Apple určené pro vývoj aplikací. Tímto softwarem je jakýkoliv operační systém OS X společně s XCode IDE. Dalšími nástroji určenými pro vývoj a především optimalizaci aplikací jsou například nástroje podporující Git, OpenGL extension Viewer, JSON Formatter a další. Nejvhodnějším nástrojem pro optimalizaci je aplikace Instruments, jenž poskytuje

[3] Zdroj http://www.apple.com/cz/iphone/specs.html

[4] Zdroj http://www.apple.com/cz/ipod-touch/specs.html

zabudované moduly pro optimalizace aplikací. Optimalizace a aplikace Instruments bude podrobněji popsána v následujících kapitolách.

2.2.2 Ostatní požadavky

Přestože jsou tyto požadavky označeny jako ostatní, jsou neméně důležité. Jedním z těchto požadavků je znalost SDK společnosti Apple pro vývoj na mobilních platformách. Toto SDK je velmi rozsáhlé a poskytuje nedozírné možnosti vývoje. Samozřejmě není možno se orientovat ve všech knihovnách poskytovaných tímto SDK, k tomuto účelu společnost Apple poskytuje dokumentaci, jenž je dostupná online na webu společnosti *(http://developer.apple.com)* nebo offline za pomocí IDE pod zkratkou *cmd + shift + ?* nebo pod záložkou *Help -> Developer Documentation.* Z mých dosavadních zkušeností hodnotím tuto dokumentaci jako jednu z nejlepších, se kterými jsem se prozatím setkal, jelikož obsahuje velmi podrobné a přehledné popisy API a navíc obsahuje spousty ukázkových kódů ještě více osvětlujících danou problematiku.

Nevyhnutelným požadavkem je znalost angličtiny. Tento jazyk je základem pro vývoj obecně a pro platformu iOS toto pravidlo platí dvojnásob. Veškerá dokumentace zmíněná v předcházejícím odstavci je psána anglicky a jakékoliv jiné zdroje jsou z 99% uváděny v anglickém jazyce.

Nevyhnutelnou se stává v současné době znalost jazyka Objective-C, jenž je určen jako hlavní programovací jazyk pro vývoj iOS aplikací. IDE sice podporuje integraci dalších programovacích jazyků, ovšem tyto možnosti jsou velmi omezené. Integrovat lze pouze jazyky C a C++ a pouze vybrané knihovny těchto jazyků. Programovací jazyk Objective –C se svou strukturou velmi podobá kombinaci jazyků C a JAVA. Najdeme v něm jak struktury podobné jazyku C, tak práci s daty podobnou JAVĚ.

Dalším, teď již pouze doporučeným požadavkem, je znalost diskusních fór, kde je možno nalézt odpovědi na některé základní otázky. Velmi užitečným fórem v tomto ohledu je *www.stackoverflow.com*, kde je možno nalézt nespočet odpovědí na základní otázky ohledně vývoje.

2.2.3 Testování aplikací

Testování aplikací je velmi ošemetnou záležitostí. Společnost Apple nabízí IDE simulátory pro jednotlivá zařízení. Tyto simulátory jsou velmi věrohodnými kopiemi reálných zařízení a nabízí všechny funkce poskytované zařízeními. Ovšem výkon a hardwarová akcelerace není omezena na možnosti zařízení, nýbrž poskytuje možnosti samotného hardware na němž je aplikace spouštěna v simulátoru. Tato skutečnost způsobuje značné problémy v testování, zvláště zohledníme –li testování grafických aplikací.

2.2.4 Publikace aplikace

Pro publikaci aplikace na iOS AppStore vyžaduje společnost Apple speciální kritéria[3]. Každá aplikace prochází takzvaným review, jenž je prováděno testery společnosti Apple. Toto review trvá někdy až 2 týdny. Tento čas závisí na rozsahu aplikace a na zkušenostech developera. Pokud byl již developer v minulosti postihován odmítnutím aplikace, doba review se značně protahuje. Kritéria, jenž jsou zveřejněna na webu společnosti jsou testery přísně dodržována. Pokud aplikace nesplňuje nějaké závažné kritérium, je okamžitě „smetena ze stolu“.

Kupříkladu problém nastává ve chvíli, kdy jsou v aplikaci využity API samotné společnosti Apple, ovšem tyto API jsou označeny jako privátní, a tudíž v dokumentaci se nevyskytují. Tato skutečnost vede k okamžitému odmítnutí aplikace a je požadována náprava v kódu. Poté je aplikace znovu odeslána ke schválení a celý proces se může opakovat. Dalším velmi závažným problémem je pád aplikace během testování testery. Tato skutečnost rovněž vede k okamžitému odmítnutí aplikace. Tyto skutečnosti jsou jen jedny z mále, jenž společnost Apple kontroluje.

Z pohledu vývojáře hodnotím skutečnost takovéto kontroly jako velmi svazující. Požadavky jsou mnohdy velmi mnohoznačné a nejasné a tudíž může, často z nepochopitelných důvodů, dojít k odmítnutí aplikace. Na druhou stranu , z pohledu koncového uživatele, hodnotím tuto vlastnost jako velkou výhodu, jelikož tímto způsobem Apple zajišťuje vysokou kvalitu a spolehlivost vydaných aplikací.

3 Dostupné knihovny pro editaci 2D grafiky pro iOS

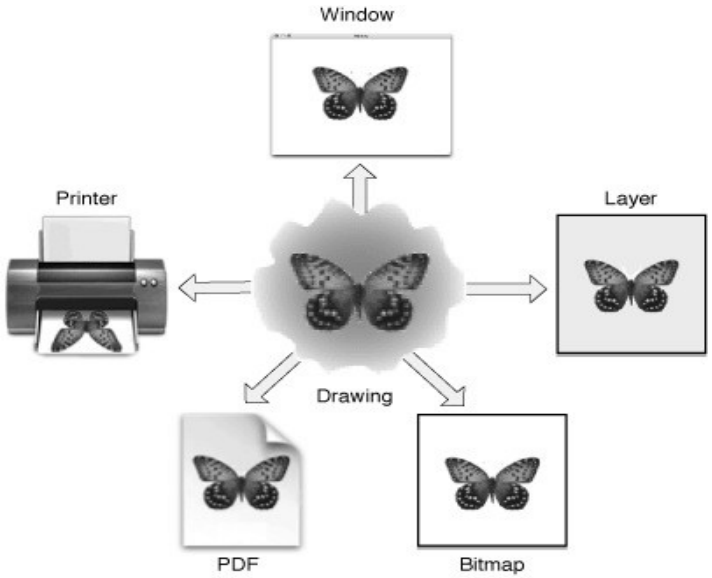
V současné době se velmi rozmáhá trend editace 2D grafiky v rámci platformy iOS. Tímto můžeme rozumět jakékoliv grafické editory či aplikace dostupné pro iOS zařízení, jenž jsou schopné jakýmkoliv způsobem pozměnit poskytnuté grafické prvky.

V následujícím bloku textu budou popsány dostupné knihovny pro vykreslování 2D grafických objektů v systému iOS. Většina dnes dostupných knihoven využívá k řešení API společnosti Apple, jenž poskytuje možnosti vykreslování jednotlivých objektů. Tímto API je myšleno Core Graphics nebo OpenGL ES.

3.1 Knihovny Core Graphics

Core graphics framework je knihovna, jenž je založena na jazyku C. Tato knihovna dle dokumentace není vnímána jako objekt, naopak pouze jako soubor funkcí sloužící pro vykreslování 2D grafických objektů. Mezi tyto objekty patří i například pohledy (View). Pro vykreslování je používán 2D rendering. Někdy je také označována, a ve straších odkazech může být nalezena, jako Quartz2D.

Nejčastějsími objekty pro vykreslování jsou kontexty, cesty, obrázky, vrstvy, vzory a mimojiné také pdf kontexty. Tato knihovna je velmi „mocná“ a obsahuje spoustu užitečných funkcí. Základním stavebním kamenem této knihovny je kontext. Ať už se jedná o jakýkoliv kontext, bitmapový, či pdf kontext, je vždy referencován pomocí struktury `CGContextRef`.



Obrázek 4: Cíle vykreslování s využitím knihovny Quartz2D[5]

[5] Zdroj http://developer.apple.com/library/ios/#documentation/GraphicsImaging/Conceptual/drawingwithquartz2d/dq_overview/dq_overview.html#//apple_ref/doc/uid/TP30001066-CH202-CJBIBHHB

Tato knihovna je hojně využívána programátory právě díky svým schopnostem. Díky tomu, že je psána v jazyce C, může být paměťově plně hlídána, a tím může být zajištěna plynulost a rychlost vykreslování. Vykreslování samo o sobě se děje pouze v jedné metodě, jenž musí být společně s užitím tohoto frameworku implementována. Jedná se o metodu `-(void)drawRect:(CGRect) rect`, kde parametr funkce reprezentuje oblast, ve které má být daný kontext vykreslen. Dokumentační soubory výhradně doporučují, aby všechen kód pro „kreslení“ byl umístěn v této metodě. Pokud zde kód nebude umístěn, může dojít k tomu, že nedosáhneme požadovaného výsledku, tedy dojde k modifikaci vykreslovaného kontextu nebo k jeho úplné deformaci.

Při vykreslování objektu je postup vždy následující. Nejprve je potřeba daný kontext získat a uložit jej, referencovat, do struktury typu `CGContextRef`. Dále následuje modifikace dle požadavků. Tyto požadavky mohou být různého typu. Může se jednat jak o interakci uživatele, tak modifikaci uvnitř kódu.

Interakcí uživatele rozumíme jakoukoliv modifikaci kontextu vyvolanou na popud uživatele. Za tuto modifikaci můžeme krom změny tloušťky, barvy, stylu a antialiasingu považovat i rotaci zařízení do jiné polohy, čímž dojde ke změně vykreslovací oblasti.

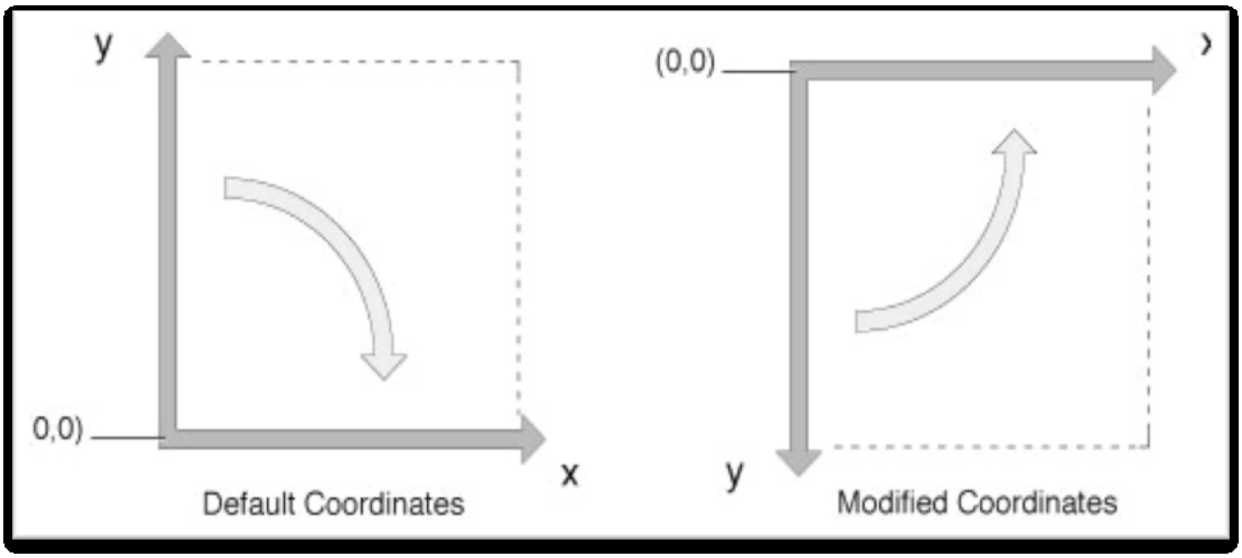
Kontext samotný v sobě dokáže uchovávat několik stavů. Tyto stavy jsou také hojně využívány, jelikož pokud dojde k chybě vykreslování, lze se vrátit k předcházejícímu stavu. Tyto stavy jsou ukládány do zásobníku, ze kterého mohou být také vyvolávány zpět. K těmto procesům slouží funkce CGContextSaveGState a CGContextRestoreGState.

Krom všeobecných informací o kontextu samotném jsou v této struktuře uchovávány i jiné vlastnosti. Výběr nejpodstatnějších vlastností `CGContextRef`:

- Barva, barevný prostor, stín a také výplň
- Využití průhledných vrstev
- Gradient
- CTM rotace
- Vykreslovací moduly
- Šířku cesty, linky
- Interpolaci
- Antialiasing
- Zploštění linky, typ linky, spoj linky a také ukončení linky
- Vzory linky (plně, čárkovaně, čerchovaně, tečkovaně)

Velkou pomocí pro rozpoznání účelu funkcí může být samotné pojmenovávání metod tohoto frameworku. Toto pojmenování je uváděno striktně v tomto tvaru: `<Typ><Akce>(<Objekt>,<Parametry>).` Příkladem takovéto funkce může být CGPathAddLineToPoint(path, nil,22.0f,100.0f). Tato funkce provede přidání linky k dané cestě ze stávajícího bodu do bodu x,y [22.0f, 100.0f]. Tento příklad byl převzat z [6].

Další velmi důležitou skutečností je určování počátku souřadné soustavy ve frameworku Quartz2D. Core Graphics počítá se souřadnou soustavou, jenž svůj počátek umisťuje do levého dolního rohu obrazovky bez ohledu na orientaci zařízení.



Obrázek 5: Systém koordinátů pro Quart2D[6]

Tento text a úryvky byly převzaty z [7].

3.1.1 Knihovna Animated Paths

Tento velmi jednoduchý framework využívá pro vykreslování objektů knihovnu `QuartzCore`. Tato knihovna je zahrnuta ve frameworku `Core Graphics` a využívá k vykreslování 2D renderovací techniky. Knihovna Animated Paths je publikována pod licencí MIT.

Tato knihovna není přímo využitelná pro editaci 2D grafiky, avšak poskytuje možný náhled na řešení daných problémů. Obsahuje velmi málo funkcí, avšak hlavní devízou této knihovny je ukázka spojení `Core Animation` společně s `Core Graphics`. Tyto dva frameworky jsou velmi často využívány společně. `Core Graphics` jako samostatná vykreslovací jednotka poskytuje vysoký výkon a spolehlivost aplikace a `Core Animation framework` poskytuje velmi rychlé a spolehlivé řešení pro animování objektů. Tento projekt se zaměřuje na vykreslování předem definovaných textových řetězců a také na vykreslování předem definovaných objektů. Renderovacím plátnem je v této knihovně `CALayer`, jenž poskytuje rozsáhlé možnosti pro animace.

`CALayer` dědí své vlastnosti z třídy typu `NSObject`, což definuje jeho vlastnosti v rámci `Cocoa Touch frameworku`.

Výhody :

- Spojení `Core Animation` a `Core Graphics`
- Zdrojový kód je poskytován pod licencí MIT

[6] Zdroj https://developer.apple.com/library/mac/documentation/GraphicsImaging/Conceptual/drawingwithquartz2d/Art/flipped_coordinates.jpg

Nevýhody:

- Nulová podpora vykreslování 2D objektů
- Nulové udržování frameworku
- Žádná dostupná rozšíření

Tento framework je dostupný na githubu zde [8].

3.1.2 Knihovna Paint Pad

Tato aplikace využívala knihovnu PaintiOS, jenž byla volně dostupná pod GPL licencí. Tato knihovna však byla v průběhu psaní této práce odebrána. Nicméně tato aplikace využívá část již zmiňované knihovny, společně s částečnými modifikacemi. Základním stavebním kamenem této aplikace a zmiňované knihovny je třída `QuartzFunView`. Tato třída obsahuje většinu logiky potřebné pro vykreslování objektů za pomocí `Core Graphics` funkcí. Hlavní funkcí pohledu `QuartzFun` je `saveCurrentViewToPicture` jenž ukládá samotný kontext vyrenderovaný do PNG či JPEG formátu pro pozdější použití. Tento pohled také reaguje na interakci uživatele pro vykreslování dotyků na obrazovce. Tyto dotyky jsou zobrazovány jako kontinuální cesty nebo body. V tomto případě záleží na typu dotyku. V rámci této aplikace není uveden typ licence, tudíž není možno posoudit, zda může být použita pro vývoj nebo ne. Nicméně obrovskou nevýhodou této aplikace je nulová dokumentace a také to, že komentáře jsou uvedeny pouze v čínštině.

Výhody:

- Ukázka použití Core Graphics renderování
- Vysoká variabilita renderovaných objektů (Elipsa, Kruh, Linka, Bitmapa)
- Variace typu zobrazení linky
- Samostatný výběr barvy za pomocí pole a variace samotných RGB hodnot

Nevýhody:

- Nulová dokumentace
- Komentáře pouze v čínštině a minimálně v angličtině
- Nedostatečné rychlost zobrazování na zařízeních
- Zahlcování renderovacích schopností zařízení
- Uživatelsky velmi složitý a programátorsky nepřehledný

Tato aplikace a její zdrojové kódy jsou volně dostupné na [9].

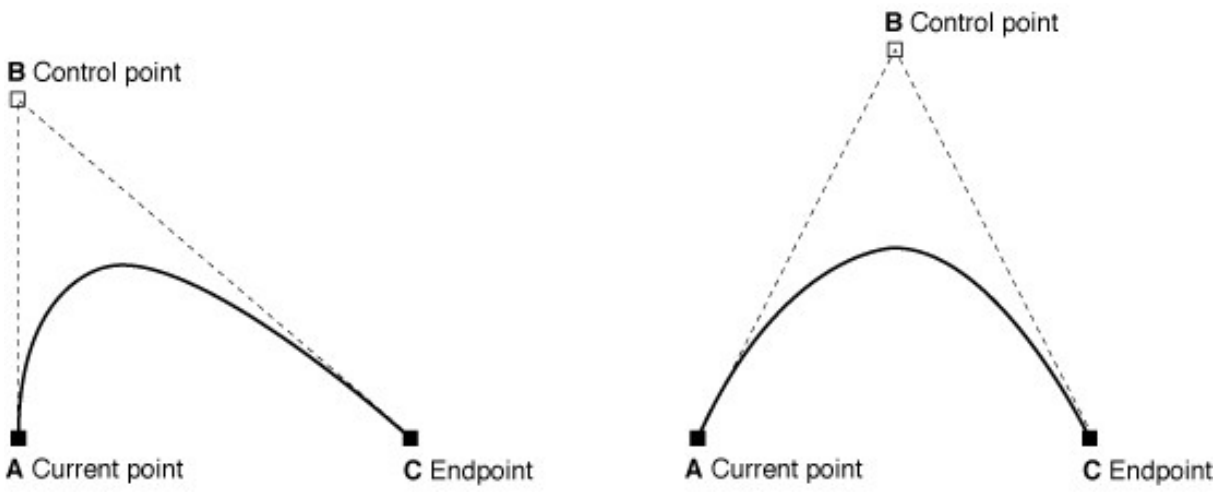
3.1.3 Knihovna Smooth Line View

Cílem tohoto projektu je navrhnout pohled, jenž bude schopen generovat jemné linky za pomocí dotyků uživatele. Tyto linky by měly být generovány velmi rychle za předpokladu nevyužití více komplexních frameworků jako je například OpenGL ES. [10]

Tento projekt se zaměřuje především na výkonnost aplikace a neobsahuje jiné funkce než pouhé vykreslování linky za pomocí interakce uživatele. Velkým pozitivem tohoto projektu je využití

pouze Core Graphics funkcí s tím, že pro vykreslování samotných linek jsou využívány kvadratické křivky.

Tyto křivky poskytují velmi jemné přechody při změnách směru díky tomu, že obsahují jeden nebo i více kontrolních bodů. [11]



Obrázek 6: Kvadratické křivky[7]

Další nespornou výhodou tohoto projektu je přehlednost kódu. Při testování na reálných zařízeních jsme tento projekt vyhodnotil jako nejúspěšnější z důvodu rychlosti a spolehlivosti vykreslování linek. Při testování na displejích s vysokým rozlišením (iPad retina, iPhone5) byla tato knihovna velmi spolehlivá a rychlá.

Výhody:

- Vysoká rychlost renderování
- Čisté hrany při přechodech a rychlých změnách směru křivky
- Nízké zatížení GPU
- Funkčnost na retina displejích

Nevýhody:

- Velmi nízká funkčnost (jedná se pouze o vykreslování linek)
- Chyby při rotaci zařízení
- Nevyřešená správa paměti při vyšším zatížení zařízení

[7] Zdroj http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIBezierPath_class/Reference/Reference.html#//apple_ref/occ/instm/UIBezierPath/addQuadCurveToPoint:controlPoint:

3.1.4 Knihovna Cloud finger Paint

Cloud finger paint je projektem jenž má své kořeny v Asii. Jedná se o velmi jednoduchý modul, který lze použít v případě požadavků odeslání vytvořené 2D grafiky tiskárně.

Tento modul poskytuje zdrojové kódy pro iOS aplikaci, serverovou část, část kódu pro server, který obsluhuje tiskárnu a také dokumentaci, jak tento projekt použít.

Skýtá však možná úskalí v podobě použití. Při bližším nahlédnutí do zdrojových kódů jsem zjistil, že možnosti změn, například tloušťky linky a podobných vlastností `CGContextRef`, jsou velmi omezené nebo žádné.

Kód aplikace hodnotím jako velmi neefektivní z důvodů, že nenásleduje doporučení společnosti Apple. Dle dokumentace by měl být všechen kód určený pro vykreslování umístěn v metodě `drawRect:`, ovšem tento modul toto doporučení ignoruje.

Všechen kód, který se stará o vykreslování, je umístěn v reakčních metodách frameworku CocoaTouch. Tyto metody jsou dle dokumentace volány i v případech, kdy je zjištěna interakce uživatele s dotykovou plochou mimo oblast určenou pro generování 2D grafiky, proto může docházet ke zbytečnému zatěžování zařízení i ve chvílích, kdy to není vyžadováno.

Důkazem tohoto faktu může být testování aplikace na reálném zařízení. Dle dokumentace by měla být reakční doba dotyku obrazovky nižší než 0.2 s, ovšem v případě testování této aplikace se jednalo až o 1.0 sekund.

Zdrojové kódy projektu jsou dostupné online na serveru GitHub zde [12]. Ukázkové video aplikace je dostupné online na serveru YouTube společně s odkazy na moduly pro server side aplikace. [13]

Výhody:

- Modul přináší možnost odeslat vytvořenou 2D grafiku tiskovému serveru
- Jednoduchost a přehlednost

Nevýhody:

- Nesprávná implementace dle Quartz2D Programming Guide
- Všechen kód je „nacpán“ do jedné třídy
- Neefektivnost implementace
- Nulové možnosti změny CGContextRef pomocí přístupových metod
- Vícenásobné vytváření kontextů

3.1.5 Shrnutí Core Graphics

Core graphics, neboli Quartz2D, je velmi mocným frameworkem, jenž nabízí velmi rozsáhlé možnosti editace 2D grafiky. Výše uvedené frameworky využívají vždy jen nepatrnou část jeho možností. Uvedené frameworky a moduly rozsáhle využívají možnosti vykreslování dotyků uživatele. Žádný z těchto modulů však neposkytuje podporu více dotyků. Dalšími problémy jsou žádná nebo nedostačující dokumentace, neuvedená licence a především nulová možnost změny parametrů

struktury `CGContextRef`. Tyto problémy by bylo vcelku jednoduché vyřešit, pokud by se autoři chtěli více věnovat rozšíření svých knihoven.

Smooth Line View hodnotím jako nejlepší a nejvýkonnější modul. Tento modul sice obsahuje pouze algoritmus pro vykreslování interakcí uživatele a dále neposkytuje žádné možnosti editace 2D grafiky, ovšem potencionálně by mohl, díky své výkonnosti, tvořit základ pro implementaci mého řešení. Ostatní moduly jsou výkonnostně chabé a jedinou zajímavostí je spojení tiskového serveru s iOS aplikací v rámci Cloud Finger Paint knihovny.

Placené knihovny a frameworky neuvádím z důvodu, že jsem žádné nenalezl v době tvorby této práce.

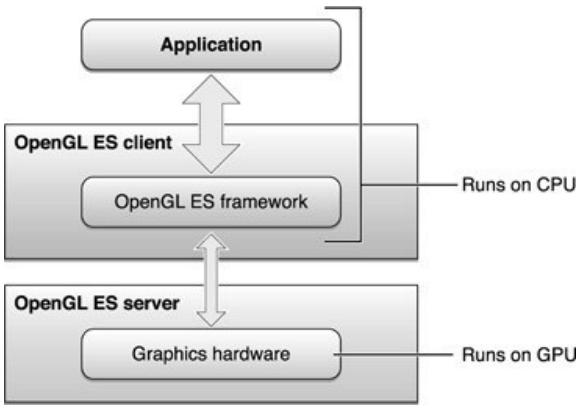
3.2 Knihovny OpenGL ES

Platforma iOS nabízí další možnost zobrazování. A to OpenGL ES. Tato technologie je multiplatformní a běžně se používá i pro desktopové aplikace díky své podpoře 2D a 3D scén. Pro platformu iOS jsou dostupné dvě verze této technologie. Jedná se o verze OpenGL ES 1.1 a také OpenGL ES 2.0, která je v dnešní době využívána nejvíce.

OpenGL ES je soubor funkcí napsaných v jazyce C a je označováno jako neurální API. Tyto funkce mohou být hardwarově akcelerovatelné přímo na grafické kartě zařízení, a proto se OpenGL ES využívá především pro projekty, jenž jsou velmi náročné na výkon zařízení. V těchto případech můžeme hovořit o 3D hrách. Příkladem takovéto hry může být veleúspěšná hra Temple Run. [23]

3.2.1 Architektura OpenGL ES

OpenGL ES používá architekturu klient – server. Tato architektura je využívána speciálně pro hardwarovou akceleraci. OpenGL ES je simplifikací OpenGL. Toto zjednodušení přináší zjednodušení a odebrání přebytečné funkcionality OpenGL. Důvodem pro tento krok je snadnější orientace v API a jeho implementace v rámci mobilních zařízení.



Obrázek 7: Architektura OpenGL ES[8]

[8] Zdroj http://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/Art/cpu_gpu.jpg

Rozložení aplikace a její implementace se různí pro verzi OpenGL ES a také je odlišná pro jednotlivé platformy. Dostupnými platformami pro tuto technologii jsou Mac OS X a také iOS. Standart nedefinuje funkce pro spolupráci s grafickým systémem. Tyto požadavky jsou řešeny vždy unikátně pro každou jednotlivou platformu. Proto musí vždy být poskytovány implementace pro framebuffery a renderovací kontexty.

Renderovací kontext

Toto označení se používá pro vnitřní stav aplikace, konkrétně tento kontext zobrazuje stav frame bufferů.

Systémové framebuffery

Buffery uchovávající informace o vykreslování pomocí OpenGL ES. V rámci iOS se nejedná o systémové framebuffery, protože ty nejsou defaultně poskytovány. Namísto toho se jedná o objekty tohoto typu jenž jsou alokovány pomocí OpenGL ES API tak, že jejich obsah může být poskytován `Core Animation` frameworku, který jako jediný slouží pro vykreslování window kontextu v rámci iOS.

3.2.2 Problémy OpenGL ES

Jelikož je Open GL ES velmi výkonné a poskytuje navíc hardwarovou akceleraci, mohlo by se jednat o technologii velmi vhodnou pro editaci 2D grafiky. Opak je však pravdou, jelikož výkonnost není vše, co musí být zohledněno při tvorbě takovéhoto frameworku. Je pravdou, že se velmi hodí pro zobrazování 3D grafických scén s velmi propracovanou a grafikou, ovšem je již méně vhodný pro zobrazování 2D grafiky vytvářené například za pomocí interakcí uživatele.

Open GL ES defaultně neposkytuje reakční metody jenž zachytávají interakci uživatele tak, že by poskytovaly více informací než jen dotykový bod. Tato skutečnost je velmi podstatná pro budoucí implementaci. Dalším zásadním problémem je nutnost vytváření framebufferů a kontextů pro vykreslování. S využitím Cocoa Touch frameworku tyto kontexty aplikaci znatelně zpomalují.

Popis architektury OpenGL ES byl převzat z [10].

3.2.3 OpenGL ES Drawing guide

Základním stavebním kamenem pro vykreslování pomocí OpenGL ES je framebuffer. Dalšími potřebnými faktory pro správné vykreslení grafického objektu jsou color buffer, depth neboli stencil buffer a také textura, která má být pro daný objekt vykreslena.

Kroky pro vytvoření framebufferu:

- Vytvoření a přiřazení framebufferu

Výpis kódu 1: Vytvoření a přiřazení framebufferu

```
GLuint framebuffer;
glGenFrameBuffers(1,&framebuffer);
glBindFrameBuffer(GL_FRAMEBUFFER, framebuffer);
```

- Vytvoření renderbufferu a jeho přiřazení framebufferu

Výpis kódu 2: *Vytvoření renderbufferu a jeho přiřazení framebufferu*

```
Gluint colorRenderbuffer;
glGenRenderbuffers(1,&colorRenderbuffer);
glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);
glRenderbufferstorage(GL_RENDERBUFFER,GL_RGBA8,width,he
ight);
glFrameBufferRenderBuffer(GL_FRAMEBUFFER,GL_COLOR_ATTAC
HMENT0,GL_RENDERBUFFER, colorRenderbuffer);
```

- Vytvoření depth nebo depth/stencil bufferu

Výpis kódu 3: *Vytvoření depthbufferu*

```
Gluint depthRenderbuffer;
glGenRenderbufers(1,&depthRenderbuffer);
glBindRenderbuffer(GL_RENDERBUFFER, depthRenderbuffer);
glRendebufferStorage(GL_RENDERBUFFER,GL_DEPTH_COMPONENT
16,width,height);
glFramebufferRenderbuffer(GL_GRAMEBUFFER,GL_DEPTH_ATTAC
HMENT,GL_RENDERBUFFER,depthRenderbuffer);
```

- Otestování komplexnosti a korektnosti objektu framebufferu

Výpis kódu 4: *Otestování korektnosti framebufferu*

```
Glenum status =
glCheckFramebufferStatus(GL_FRAMEBUFFER);
if(status != GL_FRAMEBUFFER_COMPLETE){
NSLog(@"Failed to acomplish frame buffer");
}
```

Tyto 4 kroky jsou nezbytné pro korektní vytvoření objektu framebufferu. Po těchto krocích mohou následovat operace jako bindování textury a renderování framebufferu do vrstvy.

V následujícím textu budou představeny nejpoužívanější knihovny, jenž využívají OpenGL ES technologie.

3.2.4 **Knihovna Ceed GL OpenGL library**

Ceed GL OpenGL library (dále jen Ceed GL) je knihovna jenž poskytuje podporu OpenGL v iOS i Mac OS X projektech. Tato knihovna obsahuje rozsáhlé možnosti vykreslování 2D objektů. Hlavní výhodou této knihovny je enkapsulace OpenGL objektů do Objective-C objektů. Tato enkapsulace značně ulehčuje práci programátorům, jelikož většinou jsou OpenGL objekty

reprezentovány integery. Ceed GL tím, že enkapsuluje OpenGL objekty do objektů Objective –C, také mapuje vztahy mezi jednotlivými objekty a umožňuje využívat přístupové metody pro změnu jednotlivých atributů. Tímto se velmi ulehčuje manipulace s GL daty v rámci aplikace a její struktury.

Další výhodou knihovny, jak sám autor uvádí, je podpora takzvané „draw“ funkce. Tato funkce propojuje vertex buffery, textury a shadery v primitivní, znovupoužitelné objekty. Tato funkce také umožňuje snadnější přístup k maticím a atributům shaderů.

CeedGL je poskytována v rámci 3-clause BSD licence a přímo využívá OpenGL ES 2.0.

Výhody:

- Ceed GL využívá OpenGL ES 2.0
- Definice „draw“ funkce s poskytnutím přístupových metod pro atributy shaderů
- Enkapsulace OpenGL objektů
- Poskytování pod 3-clause BSD licencí
- Vysoký výkon a rychlost vykreslování
- Návod pro implementaci CeedGL ve vlastním projektu
- Ukázkové kódy

Nevýhody:

- Žádná dokumentace API
- Chyby při prvotní kompilaci, nutnost oprav poskytovaného kódu

Tato knihovna je dostupná online.[14]

3.2.5 Knihovna Smooth drawing

Knihovna Smooth drawing poskytuje API pro vykreslování jemných linek. Neposkytuje však enkapsulaci jako knihovna CeedGL a tudíž je její použití mírně komplikovanější. Integrace knihovny je možná pouze v rámci iOS. Pro zkušenějšího programátora je integrace této knihovny velmi jednoduchá, jelikož je umístěna pouze v jedné třídě. Autor však uvádí, že umístění do jedné třídy bylo provedeno za účelem možnosti využití ARC, jenž bylo uvedeno v rámci iOS 5.

Na druhou stranu knihovna neposkytuje skoro žádné přístupové API pro změny atributů. Toto API by se však dalo jednoduše vytvořit vzhledem k tomu, že celá knihovna je uložena pouze v jedné třídě. Ovšem tato skutečnost přispívá i k jisté nepřehlednosti a složité orientaci v kódu.

Výhody:

- Jednoduchá integrace v rámci projektů určených pro platformu iOS
- Velmi „jemné“ a přesné vykreslování linek
- Integrace společně s frameworkem Cocoa Touch

Nevýhody:

- Téměř žádná dokumentace
- Potencionální rozšíření by bylo velmi náročné
- Využití GestureRecognizerů namísto CocoaTouch

Smooth drawing je dostupná online na serveru GitHub. [15]

3.2.6 Framework Sparow[16]

Tento framework je založen na technologii OpenGL ES. Většina jeho knihoven je psána pomocí OpenGL s využitím knihoven, které jsou nativně poskytovány systémem iOS. Tento framework je jedním z nejrozsáhlejších dostupných na trhu iOS. Obrovskou výhodou je to, že je poskytován zdarma v rámci zjednodušené BSD licence. Tento framework je perfektně zdokumentován a dokonce je k tomuto frameworku založeno fórum, které je dostupné online. [16]

Za zmínku rozhodně stojí, že tento framework je designován především pro vývoj herních aplikací v rámci systému iOS, a proto je jeho výkon značně vysoký. Autoři velmi dbali na to, aby byl napsán čistě a pouze v jazyce Objective – C. Tato skutečnost značně ulehčuje práci se Sparrow.

Prvotním problémem jistě bude OpenGL, ovšem proto je přímo na oficiálních stránkách frameworku poskytována podpora [17]. Tvůrci Sparrow také poskytli výbornou dokumentaci poskytovaného API.

Donedávna nebyla editace 2D grafických objektů podporována, ovšem v rámci verze 1.1 byla do tohoto projektu přidána třída SHLine a jiné. Tato třída umožňuje vytváření linek s určitou tloušťkou, alpha kanálem a dalšími atributy. Dokumentace k tomuto rozšíření je dostupná zde [18].

Sparrow nabízí rozsáhle možnosti renderování 3D scén, ovšem s 2D grafickými objekty si donedávna poradil jen s pomocí Core Graphics a strukturou CGContextRef. Pro účely editace 2D grafiky je Sparrow prozatím nedostatečně vybaven.

Výhody:

- Robustní a velmi obsáhlý framework
- Podpora nativních knihoven iOS
- Napsán pouze v Objective-C
- Rozsáhlá podpora renderování
- Výborná podpora 3D objektů
- Založen na OpenGL ES a tudíž vysoká výkonnost
- Výborná dokumentace a fórum
- Open source framework
- Kontinuální vylepšování autory

Nevýhody:

- Malá podpora editace 2D grafiky
- Výborný pro hry, ovšem méně vhodný díky složitosti pro ostatní aplikace

3.2.7 Framework Cocos2D

Cocos2D framework (dále jen Cocos2D) je dalším ze zástupců robustních frameworků založených na technologii OpenGL ES. Teto framework je primárně určen pro vývoj herních aplikací,

ovšem v rámci verze 2.0 přináší také rozšířené možnosti pro vykreslování primitivních 2D objektů. Těmito objekty jsou například kruh, ovál, trojúhelník a jiné.

Tento framework je poskytován v rámci BSD licence a je plně zdokumentován. Velkou výhodou je fórum a také rozsáhlé návody i s ukázkovými zdrojovými kódy. Tím, že Cocos2D poskytuje vykreslování primitivních objektů značně ulehčuje práci programátorům. V rámci zveřejnění verze 2.0 byly zveřejněny i třídy, které obsluhují vykreslování 2D objektů. Tím, že je tento projekt open source a volně dostupný, mohou být třídy volně editovány. Editace do značné míry ulehčuje vylepšení poskytovaného kódu.

Ukázkou může být následující úryvek kódu, který poskytuje vykreslování definovaných linek a tím i znázorňuje použití Cocos2D.

Výpis kódu 5: Cocos2D ukázkové vykreslení linky

```
+(CCScene *) scene {
    CCScene *scene = [CCScene node];
    HelloWorldLayer *layer = [HelloWorldLayer node];
    [scene addChild: layer];
    return scene;
}
- (void)draw {
   CGSize winSize = [[CCDirector sharedDirector] winSize];
    CGPoint center = ccp(winSize.width/2,winSize.height/2);
    CGPoint left = ccp(0,center.y);
    CGPoint right = ccp(winSize.width,center.y);
    CGPoint top = ccp(center.x,winSize.height);
    CGPoint bottom1 = ccp(winSize.width/3,0);
    CGPoint bottom2 = ccp((winSize.width/3*2),0);
    glEnable(GL_LINE_SMOOTH);
    glLineWidth(3.0f);
    glColor4f(0.8, 1.0, 0.76, 1.0);
    ccDrawLine(bottom1,top);
    ccDrawLine(top, bottom2);
    ccDrawLine(bottom2, left);
    ccDrawLine(left, right);
    ccDrawLine(right, bottom1);
}
```

Úryvek přepisuje klíčovou metodu *draw,* která je v rámci Cocos2D využívána pro vykreslování stejně jako metoda *–drawRect:* v rámci `CoreGraphics`. V rámci této ukázky je názorně předvedeno využití frameworku v praxi. Výsledkem exekuce kódu bude hvězda vykreslená na obrazovce. Tento úryvek kódu byl převzat z [20].

Cocos2D poskytuje rozsáhlé možnosti editace 2D grafiky.

Stěžejními metodami pro její editaci jsou :

- Draw
- ccDrawLine
- ccDrawCircle
- ccDrawPoly
- ccDrawSolidRect

Tento framework a návod na jeho integraci je poskytován v rámci webové prezentace [23].

Výhody:

- Podpora OpenGL ES 2.0
- Napsán v jazyce Objective – C
- Jednoduchá integrace dle návodů poskytovaných online
- Rozsáhlá dokumentace pokytovaného API
- Open source řešení poskytováno zdrama v rámci BSD licence
- Velmi jednoduchá implementace rozšíření
- Podpora vykreslování 2D objektů
- Fórum
- Kontinuální updaty verzí
- Integrace Box2d a Chipmunk

Nevýhody:

- Mísení OpenGL ES a CoreGraphics
- Nevyhnutelnost práce s texturami a OpenGL ES funkcemi

3.2.8 Shrnutí OpenGL ES

OpenGL ES je technologie, která je přímo embedována v rámci iOS. V současné době je dostupná v rámci verze 2.0. OpenGL ES je hojně využívána pro vývoj herních aplikací na mobilních zařízeních, ovšem není již tak vhodná pro editaci 2D grafiky v rámci platformy iOS.

Přestože v současné době jsou poskytovány frameworky jako například Sparrow, či Cocos2D, není poskytována přímá podpora vykreslování 2D objektů. Je pravdou, že autoři Cocos2D tyto možnosti postupně zavádějí, ovšem díky tomu, že celý framework je napsán v rámci užití OpenGL ES, je nesnadné využít `CocoaTouch` framework, který poskytuje nativní podporu interakce s uživatelem v rámci reakčních metod dotyků.

V současné době je OpenGL ES převážně orientováno na 3D scény a 2D scény a jejich vykreslování je ponecháno pro knihovny, jenž jsou podporovány v rámci Quartz2D.

Přesto je nutno podotknout, že nejlepšími a nejkvalitnějšími zástupci pro editaci 2D grafiky v rámci technologie OpenGL ES jsou frameworky Cocos2D a Sparrow. V první řadě pak Cocos2D, který v současné době poskytuje i definované vykreslování primitivních 2D objektů. API frameworku je výborně zdokumentováno, a proto je jeho použití značně ulehčeno.

OpenGL ES knihovny jsou v ohledu kontinuálního vylepšování daleko před CoreGraphics knihovnami. Komunita okolo frameworků Sparrow a Cocos2D je aktivní a podílí se na kontinuálním vylepšování těchto frameworků.

3.3 Zhodnocení dostupných knihoven pro editaci 2D grafiky

V následujícím textu budou vybrány dvě knihovny, potažmo frameworky a tyto budou porovnány. Atributy, které jsme se rozhodli porovnávat jsou :

- Dostupnost knihovny / frameworku
- Licence
- Možnosti API
- Dokumentace API
- Možnost změny a vylepšení API
- Rychlost vykreslování

Kód pro měření záznam času je uveden v následujícím úryvku.

Výpis kódu 6: Kód pro měření záznamu času

```
NSDate * start = [NSDate date];
NSLog(@"time took: %f", -[start timeIntervalSinceNow]);
```

Knihovny, jenž jsme zvolili pro porovnávání jsou Smooth Line View v rámci technologie Core Graphics a Cocos2D v rámci OpenGL ES.

Tabulka.1: Testování vybraných frameworků

	Smooth line view	**Cocos2D**
Dostupnost	Online	Online
Licence	BSD	BSD
Možnosti API	Velmi omezené	Velmi rozsáhlé
Dokumentace API	Neposkytována	Online

Možnosti vylepšení	Rozsáhlé	Rozsáhlé
Rychlost vykreslování[9]	0.2 s	0.14 s

V rámci testování jsme zohlednili rychlost zařízení a jeho generaci. Při testování `CoreGraphics` frameworku byla měřena reakční doba vykreslování jednotlivých linek. Tato doba byla měřena v rámci volání reakčních metod frameworku CocoaTouch, a sice *touchesBegan:withEvent:* a *touchesEnded:withEvent:*. Průměrná reakční doba byla změřena i se samotným vykreslováním na 0.2 s.

V rámci technologie OpenGL ES jsme měřili reakční dobu aplikace od interakce uživatele po dokončení metody *draw:*. Naměřené hodnoty se v rámci jednotlivých zařízení příliš nelišily, a to z důvodu podpory OpenGL ES 2.0 již od verze iOS 5.1.x.

Smooth Line View umožňuje rozsáhlejší možnosti editace. Na druhou stranu Cocos2D je omezován tím, že je implementován pomocí OpenGL ES. CoreGraphics umožňuje editaci atributů v rámci jednoho kontextu, OpenGL umožňuje globální editaci těchto atributů.

Při testování a integraci jednotlivých technologií v rámci testování výkonnosti jsme zjistili, že pro možnosti implementace je co do složitosti jednodušší implementovat knihovny CoreGraphics oproti OpenGL. OpenGL požaduje spoustu rozšíření a frameworků poskytovaných nativně systémem iOS. Z důvodu nedostatečných informací v rámci návodů pro zabudování OpenGL technologií je složité korektně zaembedovat frameworky do již existujících, či nových projektů.

Rozšiřování jednotlivých technologií bylo prováděno pouze v rámci testování a pouze v omezeně. Rozsáhlejší implementace neměly v tuto chvíli smysl, jednalo se pouze o testování.

Knihovna Smooth line view byla rozšířena o API, které definovalo přístupové metody k atributům *CGContexRef*. Atributy, které jsme rozšiřovali pro účely testování byly šířka linky, styl linky a její barva. Integrace tohoto API byla vzhledem k přehlednosti kódu jednoduchá a přímočará.

Při rozšiřování frameworku Cocos2D jsme se nechali inspirovat již uváděnou referencí [20]. V rámci tohoto rozšíření jsme vytvořili metody pro požadavek vykreslení jednotlivých primitivních typů a linky z bodu do bodu.

Tyto body byly vytvářeny za pomocí integrace reakčních dotykových metod frameworku Cocos2D. Následující kód zobrazuje užitou implementaci.

Výpis kódu 7: Cocos2D reakční metody dotyku a vykreslování místa dotyku

```
- (void) ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent
*)event {
    UITouch *touch = [touches anyObject];
    CGPoint point = [touch locationInView: [touch view]];
```

[9] Toto testování bylo prováděno na zařízeních, jenž jsou uvedena v příloze. Čas vykreslování byl průměrován.

```
    [self registerPoint:point];
}
- (void) ccTouchesMoved:(NSSet *)touches withEvent:(UIEvent
*)event {
    UITouch *touch = [touches anyObject];
    CGPoint point = [touch locationInView: [touch view]];
    [self registerPoint:point];
}
- (void) ccTouchesEnded:(NSSet *)touches withEvent:(UIEvent
*)event {
    UITouch *touch = [touches anyObject];
    CGPoint point = [touch locationInView: [touch view]];
    [self registerPoint:point];
}
-(void)registerPoint:(CGPoint)point_to_register
{
     [points addObject:point_to_register];
}
-(void)removeAllPoints
{
     [points removeAll];
}
-(void)draw
{
glEnable(GL_LINE_SMOOTH); // define line_smoothness
     glLineWidth(3.0f); // define width
     glColor4f(1.0, 0.0, 0.0, 1.0); // define color
     for(int i =1; i <points.count - 1; i++){
          ccDrawLine(i,i+1);
}
}
```

Tato implementace sloužila pro testování vykreslování linek. Problémem implementace v tomto případě bylo vykreslování všech linek až po ukončení interakce s dotykovou oblastí.

S přihlédnutím k testování a použitým frameworkům hodnotíme Core Graphics knihovny jako vhodnější pro využití editace 2D grafiky, a to z důvodů náročnosti implementace a z důvodů malých rozdílů v rámci výkonnosti. Pokud hovoříme pouze o 2D grafice není nutné implementovat OpenGL

metody, které poskytují rozsáhlou podporu pro renderování 3D scén. Další výhodou `Core Graphics` je přímá podpora `Cocoa Touch` frameworku a také dalších Core frameworků.[10]

[10] Dalšími Core frameworky rozumíme CoreFoundation, CoreAnimation, CoreImage

4 Požadavky na vlastní řešení dané problematiky

4.1 Základní požadavky

V rámci vlastního řešení bude kladen důraz na funkčnost a především na rychlost vykreslování, tedy výkon. Prioritními funkcemi budou vykreslování jemných linek, nastavování jednotlivých parametrů, změny barvy dle definovaného výběru a v neposlední řadě integrace sdílení vytvořené grafiky v rámci vybraných sociálních sítí.

Před započetím implementace bylo rozhodnuto, že vhodnější technologií pro tuto problematiku bude technologie `Core Graphics`, která nabízí lepší API pro detekování dotykových událostí a je také přívětivější pro integraci v rámci projektu.

Jako inspirace poslouží knihovna Smooth line view, která využívá CoreGraphics funkcí. Inspirací v rámci desktopových aplikací může být například aplikace Photoshop CS5.[11]

4.1.1 Funkční požadavky

- Možnost vytváření 2D grafiky pomocí interakce uživatele
- Editace vytvořené grafiky
- Editace 2D grafiky ve formátu PNG, JPG, JPEG importované z knihovny obrázků
- Editace 2D grafiky ve formátu PNG, JPG, JPEG importované z fotoaparátu
- Modifikace specifických atributů
    - Barva
    - Tloušťka linky
    - Alfa kanál
    - Jas barvy
    - Styl linky
    - Střední omezení
    - Ukončení linky
    - Spojení linek
- Specifické funkce spojené s editací 2D grafiky
    - Kapátko
    - Guma
    - Vylévání barvou
- Vyčištění pracovní plochy
- Sdílení na sociálních sítích
- Uložení vytvořené grafiky ve formátu PNG či JPG
- Vykreslování základních tvarů
    - Kruh
    - Elipsa

[11] Reference http://www.adobe.com/cz/products/photoshopfamily.html

    - Trojúhelník
    - Čtverec
- Panel nástrojů

Další možná rozšíření mohou být zvážena a později doplněna v rámci implementace. Pokud se takováto rozšíření vyskytnout, budou později zdokumentována.

4.1.2 Nefunkční požadavky

- Vysoká rychlost vykreslování
- Přívětivé UI frameworku
- Přehledné zobrazení ovládacích prvků frameworku
- Jednoduchý způsob používání frameworku
- iOS styl frameworku a práce s daty[12]

4.1.3 Specifické požadavky

Prvním z požadavků je *překrývání barev* (mísení). Tímto rozumíme vrstvení jednotlivých barev tak, že pokud uživatel vykreslí podkladovou barvu žlutou s alfa kanálem nastaveným na maximální hodnotu a další vrstvou bude barva modrá s alfa kanálem nastaveným na poloviční hodnotu, tak výsledná barva bude zelená[24].

Druhým požadavkem je přesnost *kapátka*. Tato funkce vzhledem k nedostupnosti stylusu pro zařízení by měla být natolik přesná, že bude schopna zobrazit zvolenou barvu v místě dotyku. Tato barva bude zobrazena uživateli jako součást panelu nástrojů.

Posledním specifickým požadavkem je *vylévání barvou*. Primárně tato funkce bude sloužit pro malé plochy, ovšem musí být použitelná i pro větší plochy.Předpokládá se, že v tomto ohledu budou provedeny pozdější optimalizace.

4.1.4 Dodatečné požadavky

Za dodatečné požadavky byly stanoveny takové požadavky, které vycházejí ze zhodnocení již vytvořených knihoven. Vytvořená knihovna by měla být napsána tak, aby byla bez větších problémů znovu použitelná v rámci budoucích projektů. Její architektura by měla korespondovat s běžnými návrhovými vzory používanými v rámci iOS platformy. Dalším důležitým faktorem bude plynulé zobrazování v rámci vykreslovacích metod. Relevantním faktorem pro tuto skutečnost může být možnost a schopnost se na zařízení podepsat.

[12] Tímto stylem rozumíme použití specifických vzorů a sdílení informací, které je vlastní iOS

5 Vlastní implementace knihovny pro editaci 2D grafiky

Tato část textu bude sloužit pro popis návrhu a implementace vlastní knihovny. V této části bude detailně popsán návrh knihovny a její realizace. Budou zde rozebrány a vysvětleny problémy, kterým bylo potřeba v rámci implementace čelit. V závěru této kapitoly budou uvedeny další možnosti vykreslování 2D grafiky v rámci platformy iOS.

Optimalizaci výsledného řešení bude věnována samostatná kapitola.

5.1 Zvolená technologie

Zvolenou technologií pro vytvoření frameworku je Core Graphics. Důvod této volby je plná podpora Cocoa Touch frameworku a také podpora Core Animation. OpenGL ES nebylo zvoleno z toho důvodu, že se více hodí pro zobrazování 3D grafických scén. Core Graphics také umožňuje vyšší podporu pro interakci s uživatelem v rámci dotykových metod a *GestureRecognizer* metod.

Poučen úspěchy a problémy uvedenými v analyzovaných knihovnách se budu snažit, co možná nejvíce přiblížit implementaci optimálního řešení pro editaci 2D grafiky. Důvodem pro vlastní implementaci nové knihovny je fakt, že v rámci platformy iOS dosud neexistuje komplexní knihovna, která by umožňovala výše definované funkce pro editaci 2D grafiky pro platformu iOS.

5.2 Návrhové vzory

Nedílnou součástí implementace se staly návrhové vzory. V rámci implementace a jejího návrhu, bylo počítáno s návrhovými vzory MVC, delegate, singleton.

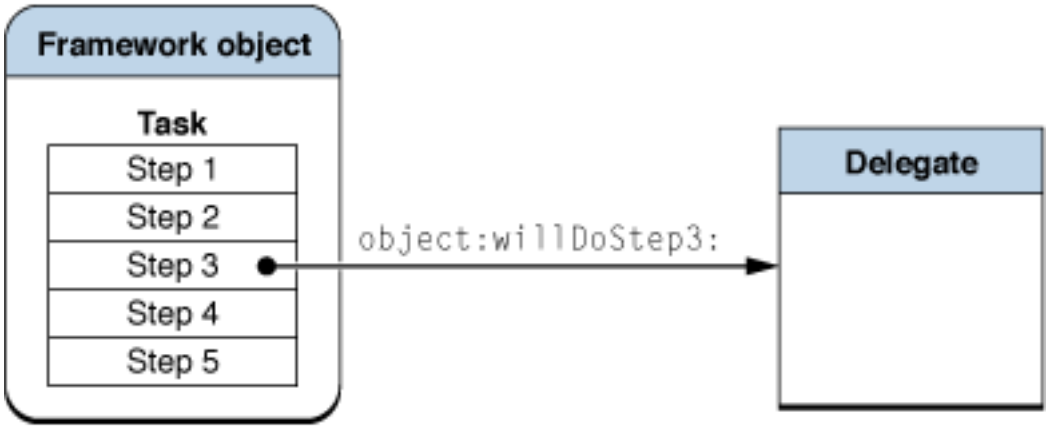
5.2.1 Vzor MVC

MVC je jedním z nejznámnějších a nejpoužívanějších návrhových vzorů.[25] Vyskytuje se cross platformě a může být aplikován v rámci mnoha různých přístupů. Velmi často je využíván frameworkem Cocoa Touch.

V rámci implementace bude tento vzorv využit tak, že třídy, jenž budou ovládat jednotlivé funkce vytvořené knihovny bychom mohli označit jako model, jelikož definují vlastnosti dané knihovny. Dále samotné jádro můžeme označit jako „view“ a za controller bude považován ViewController, který se bude starat o interakce uživatele s „view“.

5.2.2 Vzor Delegate

Delegate je jedním z návrhových vzorů, který je unikátní pro platformu iOS.[26] Jeho funkcionalitu, můžeme přirovnat k známému návrhovému vzoru Observer[27], který je však v rámci iOS prezentován jako KVO, neboli key-value-observing.[28] Funkcionalita vzoru delegát je popsána jako mechanismus, který v rámci hostovaného objektu udržuje slabou referenci na jiný objekt – delegáta tak, že v případě zaznamenání události, je delegátovi odeslána zpráva o události a delegát tuto událost přeposílá požadovaným třídám.



Obrázek 8: Delegát přijímá zprávu[13]

Delegát a jeho funkcionalita je definována jako protokol.[29] Tyto protokoly umožňují definovat metody, které budou delegátem využívány a definují také samotného delegáta.

Výpis kódu 8: MyLineDrawingView definice protokolu

```
@protocol MyLineDrawingViewDelegate <NSObject>
-(void)dropperActivatedMainThred;
-(void)check;
-(void)populateDownToolbar;
@end

id <MyLineDrawingViewDelegate> delegate;
```

5.2.3 Vzor Singleton

Návrhový vzor singleton je jedním z nejvíce využívaných vzorů pro sdílení dat v rámci aplikace.[26] Jeho výhodami jsou globální přístup a zaručení jedinečnosti v rámci aplikace. Běžně se využívá pro uložení dat, která mají být přístupná všeobecně v rámci celé aplikace.

Několik tříd frameworku Cocoa Touch využívá tento návrhový vzor. Příklady mohou být NSFileManager, UIApplication.

Návrhový vzor singleton využívá v rámci platformy iOS tzv. dispatcher.[30] Tento dispatcher zajišťuje, že daný kód bude vykonán pouze jednou. V rámci níže uvedeného zdrojového kódu můžeme vidět, že alokace sdílené instance bude provedena právě a pouze jednou.

Výpis kódu 9: Ukázka návrhového vzoru singleton

```
static FloodFillContext * sharedInstance;
```

[13] Zdroj: https://developer.apple.com/library/mac/documentation/cocoa/conceptual/CocoaFundamentals/Art/delegation.gif

```
+(id)sharedInstance
{
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sharedInstance = [[self alloc] init];
    });
    return sharedInstance;
}
- (id)init {
    if (self = [super init]) {
        [self initialSettings];
    }
    return self;
}
```

5.3 Logické a fyzické oddělení kódu

V rámci vytváření znovupoužitelné knihovny je vhodné také rozumně oddělit kód aplikace. V rámci návrhu bylo rozhodnuto, že bude odděleno samotné jádro aplikace, které se stará o vykreslování. Tomuto jádru budou přiděleny přístupové metody, které budou mít možnost modifikovat jeho vlastnosti.

Dále bude oddělen kód pro panely nástrojů, jednotlivé základní tvary, které budou v rámci knihovny definovány a také bude oddělena logika pro vyplňování oblastí. Jako samostatný projekt bude přidán modul pro sdílení v rámci sociálních sítí.

Při fyzickém rozložení kódu bude dbáno na to, aby související kód byl uveden pospolu pro lepší orientaci a přehlednost.

5.4 Automatic reference counting

ARC je novým prvkem jenž byl uveden vcelku nedávno, a to oficiálně s verzí systému iOS 5.x. Toto vylepšení umožňuje programátorům zbavit se manuálního uvolňování objektů, které bylo vždy v rámci jazyka C, ze kterého jazyk Objective-C vychází, požadováno.

Hlavním přínosem této nové funkcionality je uvolňování objektů ve chvíli, kdy doopravdy uvolněny být mohou. Lze také nově objekty přinutit k uvolnění.

Další výhodou ARC je zbavení se tzv. zombies. Zombies jsou uvolněné objekty, které však zabírají místo v paměti. Tyto objekty se stále vážou na referenci, které může, ale nemusí být systémem využívána. ARC zajišťuje, že objekty nebudou zbytečně alokovat paměť, nebudou provádět paměťové úniky (memory leaks).

Specifikace ARC dále nedoporučují vytváření struktur. Dle dokumentace je společností Apple doporučováno mapování struktur pomocí Objective-C tříd. Skutečnostmi, které by vedly k chybě kompilátoru, jsou volání metod *dealloc*, *retain*, *release*, *retainCount* a *autorelease*.

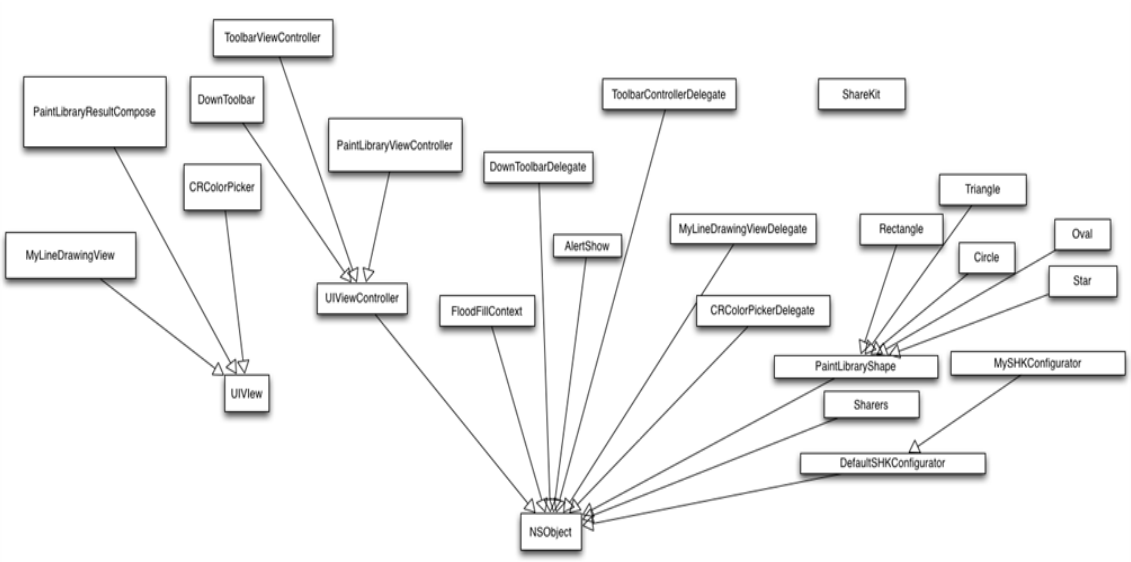
ARC může být pro lepší pochopení přirovnáno ke Garbagge Collectoru uvedeného v rámci jazyka JAVA. Ovšem funkcionalita ARC se značně a zásadně liší od GC.

Nevýhodou ARC, společně s využitím Core Graphics, je nutnost využívat uvolňujících metod Core Graphics frameworku. Toto uvolňování musí být navíc prováděno jen v rámci vybraných funkcí.

Pro využití ARC je nutno v XCode IDE přiřadit kompilátoru označení pro třídy ve tvaru *-fno-objc-arc*. [4]

5.5 Knihovna PaintLibrary

V následujícím textu bude detailně popsána vytvořená knihovna. Budou zde uvedeny funkce a implementace knihovny.



Obrázek 9: Třídní diagram PaintLibrary[14]

Na uvedeném obrázku [9] lze vidět, že celá knihovna je rozdělena do jednotlivých komponent, které řeší danou problematiku. Na obrázku lze vidět jednotlivé třídy knihovny a jejich vztahy. ORM[15] je co možná nejvíce komponováno tak, aby objekty a jejich využitelnost odpovídaly reálnému využití.

[14] Viz příloha Příloha.C:

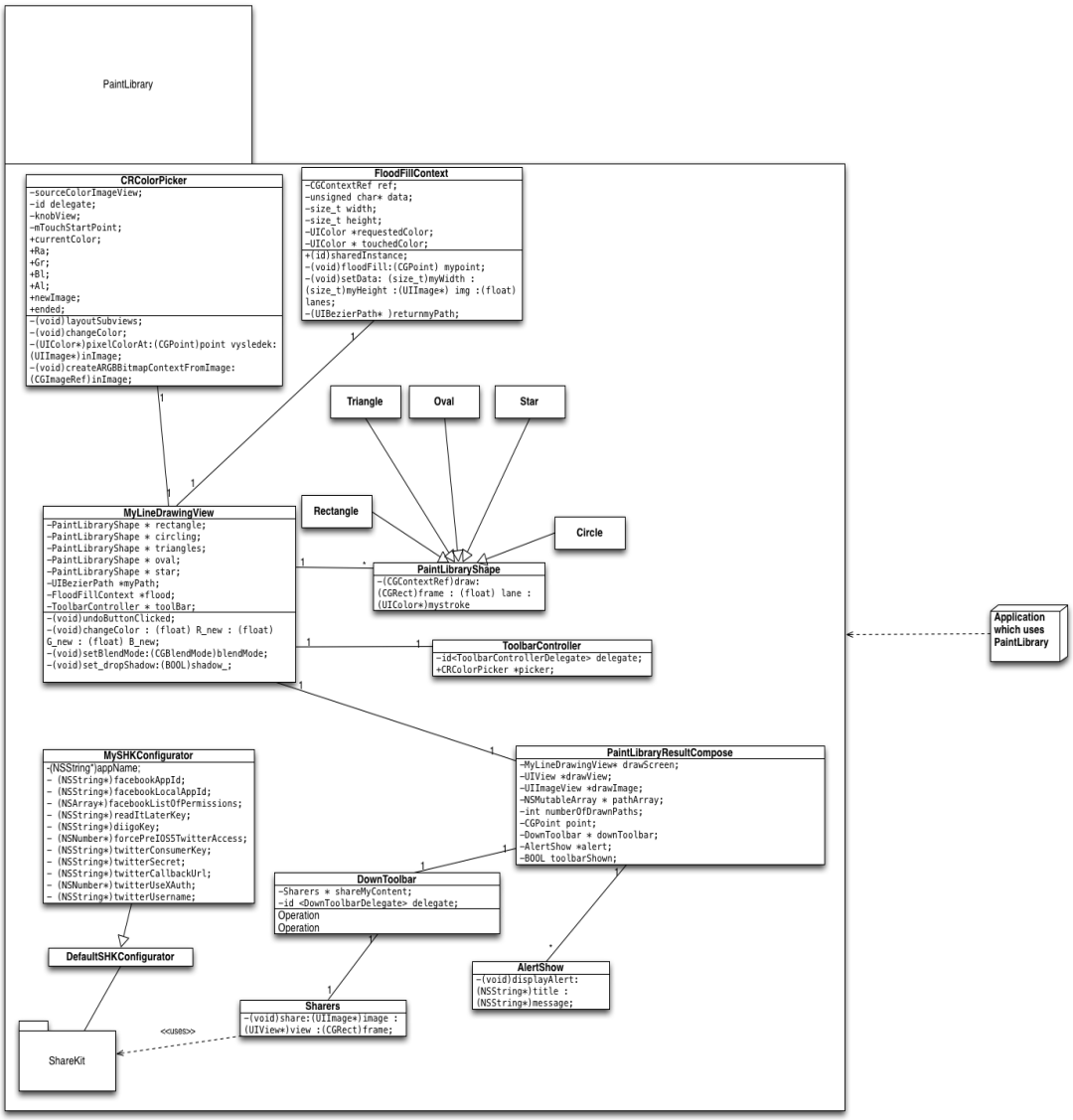
[15] ORM http://www.cs.vsb.cz/benes/vyuka/tis/prednasky/15-orm-4.pdf

Na obrázku [10] vidíme třídy a vztahy mezi těmito třídami. Jsou zde znázorněny nejdůležitější části knihovny. V rámci tohoto UML diagramu jsou zobrazeny vybrané komponenty poskytované knihovny PaintLibrary. Knihovna je komponována jako celek, aby mohla být použita v dále vyvíjených aplikacích jako samostatně stojící řešení. Stěžejními třídami knihovny jsou třídy `MyLineDrawingView`, `DownToolbar`, `ToolbarController` a `PaintLibraryResultCompose`.

`PaintLibraryResultCompose` je třída, která je stěžejní pro tuto knihovnu.

Výpis kódu 10: Instanciace PaintLibrary

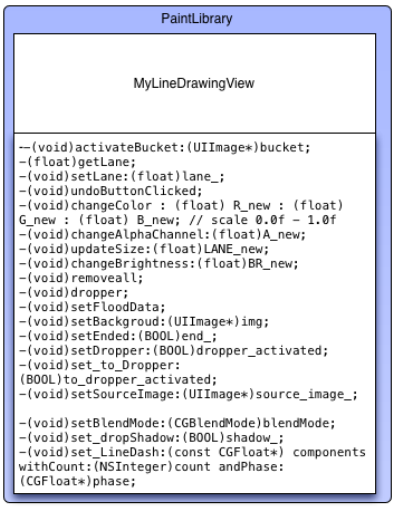
```
 if(UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone){
        drawing = [[PaintLibraryResultCompose alloc]
initWithFrame:CGRectMake(0, 0, 480, 320)];
    }
    else{
        drawing = [[PaintLibraryResultCompose alloc]
initWithFrame:CGRectMake(0, 0, 1024, 768)];
    }
    [self.view addSubview:drawing];
```



Obrázek 10: PaintLibrary UML

5.5.1 **Jádro knihovny**

Jádrem knihovny rozumíme třídu *MyLineDrawingView*. Tato třída hierarchicky dědící ze třídy UIView, která je součástí UIKit je hlavní třídou celé knihovny. Toto jádro se stará o základní funkce vykreslování a rozpoznávání dotykových metod uživatele. Důvodem pro spojení těchto dvou funkcionalit je udržení konzistentního kódu pro danou funkcionalitu.



Obrázek 11: Třídní diagram PaintLibrary – MyLineDrawingView

Jádro využívá `Core Graphics` frameworku a jeho vykreslovacích metod. Společně s vykreslovacími metodami implementuje také metody `Cocoa Touch` frameworku. Jedná se o metody *touchesBegan:, touchesMoved:* a *touchesEnded:*. V těchto metodách se jádro knihovny stará o zaznamenávání dotyků uživatele na kreslícím plátně.

Kreslícím plátnem rozumíme plochu, v rámci které jsou povoleny dotyky uživatele a na které jsou vykreslovány údaje o dotycích se speciálními vlastnostmi definovanými pomocí jádra knihovny.

Za pomocí Cocoa Touch frameworku jsou knihovnou zachycovány dotyky tak, že v metodě *touchesBegan*: je zaznamenán začátek dotyku a v rámci obslužné třídy, v tomto případě mluvíme o třídě *MyLineDrawingView* je uložen bod dotyku. Tento bod dotyku je poté přidán do kolekce bodů, které jsou spojeny ve výsledném vykreslení linkou.

Výsledná cesta, která bude vytvořena z bodů je definována typem `UIBezierPath`.[31] `UIBezierPath` je typ, který dědí z `NSObject` a umožňuje definovat geometrii, která se skládá buď z přímých, obloukovitých nebo jiných cest a tvarů. Tento objekt umožňuje obsažení mnoho segmentů. Atributy tohoto objektu mohou být přiřazovány a modifikovány separátně. Hlavní výhodou `UIBezierPath` je fakt, že mohou být, a často jsou, znovu využívány v rámci jednoho kódu. Stačí pouze cestu vyčistit od všech bodů a přesunout její začátek na počáteční bod. V našem případě hovoříme o počátečním bodě dotyku. Vyčištění cesty provedeme zavoláním instanční metody `removeAllPoints`.

Hlavním důvodem pro využití Beziérových cest je možnost kontroly křivky a spojení jednotlivých cest. Další výhodou a důvodem použití je fakt, že není nutné tento objekt alokovat znovu a zabírat tak, vcelku zbytečně, paměť, která může být využita pro výsledné vykreslování.

5.5.1.1 *Vytvoření UIBezierPath*

Vytváření se provádí pomocí operátorů `alloc` a `init`. Tyto operátory slouží k vymezení místa v paměti a k přiřazení paměti danému objektu a nastavení jeho vlastností na defaultní hodnoty.

Výpis kódu 11: Vytvoření UIBezierPath

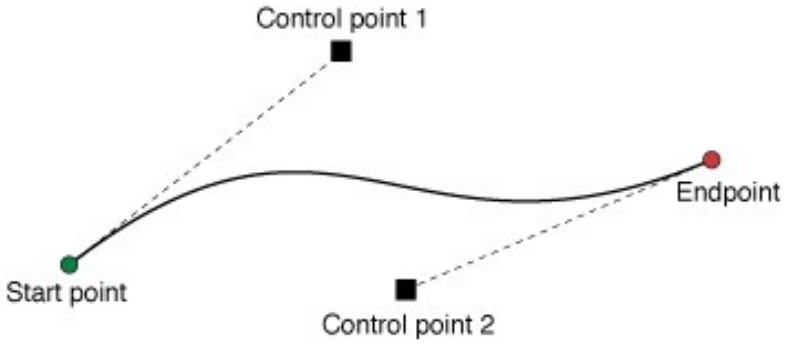
```
UIBezierPath *myPath=[[UIBezierPath alloc]init];
myPath.lineCapStyle = kCGLineCapRound;
myPath.miterLimit = 25;
myPath.lineJoinStyle = kCGLineJoinRound;
```

Instance typu `UIBezierPath` má následující vlastnosti, které je možno měnit:

- lineCapStyle – tvar cesty při jejím zakončení
- flatness – zploštění linky
- lineJoinStyle – typ spojení mezi linkami
- lineWidth – šířka linky
- miterLimit – Tato vlastnost je velmi zajímavá, protože napomáhá zalomení linky ve zlomech. Defaultní hodnota limitu je 10. Tato hodnota umožňuje plynulé „zlomení“ a přechod.

Vytvoření instance však pro vykreslování linek za pomocí dotyku uživatele nestačí. Společně s vytvořením `UIBezierPath` je nutno zachytávat dotyky uživatele, k čemuž velkou mírou napomáhá `CocoaTouch` framework se svými reakčními metodami. V těchto metodách je možno zachytávat dotyky uživatele v oblasti kreslící plochy a zaznamenávat tyto body za pomocí vytvořené cesty.

Přidávání jednotlivých bodů dotyku se provádí pomocí instančních metod `moveToPoint:` a `addLineToPoint:`. Obě tyto metody přijímají argument typu struktury `CGPoint`.



Obrázek 12: Kubická beziérova křivka[16]

Tyto metody jsou těmi nejdůležitějšími v ohledu vytváření Beziérových cest. V následujícím výpisu kódu lze vidět integrace těchto cest společně s frameworkem `Cocoa Touch`.

[16] Zdroj http://developer.apple.com/library/ios/documentation/uikit/reference/UIBezierPath_class/Art/uibezier_curve.jpg

Výpis kódu 12: Reakční metody CocoaTouch v PaintLibrary

```
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
      touch = [[touches allObjects] objectAtIndex:0];
      point = [touch locationInView:self];
      [myPath moveToPoint:[touch locationInView:self]];
      [myPath addLineToPoint:point];
}
-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
      touch=[[touches allObjects] objectAtIndex:0];
      point = [touch locationInView:self];
      [myPath addLineToPoint:point];
      [myPath moveToPoint:[touch locationInView:self]];
      [self setNeedsDisplay];
}
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
      touch=[[touches allObjects] objectAtIndex:0];
      point = [touch locationInView:self];
      [myPath addLineToPoint:point];
      [myPath moveToPoint:[touch locationInView:self]];

      [self setNeedsDisplay];
      [myPath removeAllPoints];
}
```

Ve výpisu kódu [11] je patrné, že ve všech reakčních metodách dochází k zaznamenání dotyku v pohledu, jeho převedení na strukturu typu `CGPoint` a toto struktura je poskytnuta cestě, která následně tento bod převezme a přidá jej jako další konstrukční bod cesty. Tato akce probíhá v metodách `touchesBegan:` a `touchesMoved:`. Naopak v metodě `touchesEnded:` je cesta ukončena posledním bodem a je vynuceno vykreslení dané cesty, které dle Printing and Drawing Guide iOS má probíhat v metodě `drawRect:`.

Toto doporučení je stanoveno hned z několika důvodů. Jedním z nich je konzistence kódu a druhým, neméně důležitým, je konzistence vykreslování. Tímto myslíme fakt, že pokud by kód nebyl umístěn v této metodě, mohlo by dojít k deformacím v podobě neočekávaných transformací a nebo dokonce k nezobrazení výsledného renderovaného kontextu.

Ve vykreslovací metodě se doporučuje nastavovat speciální atributy pro daný kontext, který bude renderován. Těmito atributy mohou být například oblast vykreslování, barva vykreslovaného objektu, oblast připnutí kontextu a v neposlední řadě také módy překryvu. Platforma iOS a Core Graphics podporují nativně širokou škálu překryvných módů. Více o těchto módech je uvedeno v online dokumentu Quartz2D Programming Guide.[17] Tento online dokument nabízí ukázky většiny standartních překryvných modulů a detailně popisuje jejich funkčnost.

V metodě `drawRect:` se výrazně nedoporučuje provádět složité výpočty či animace. Tento fakt by mohl vést ke značnému zpomalení vykreslovacího procesu a při velkém zatížení zařízení by mohlo dojít k pádu software. Dalším důvodem pro značné zpomalení vykreslování může být nedodržení směrnic pro vykreslování 2D grafiky poskytnutých společností Apple.

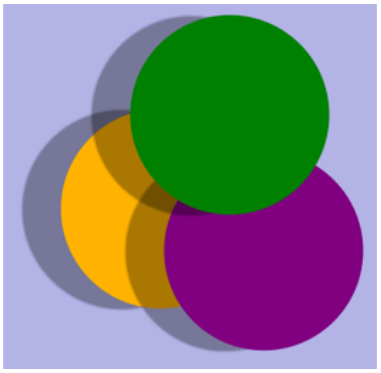
Výpis kódu 13: DrawRect metoda PaintLibrary

```
- (void)drawRect:(CGRect)rect
{
    myStroke = [UIColor colorWithRed:R green:G blue:B alpha:A];

    if (lane_bool) {
        [myStroke setStroke];
        [myPath strokeWithBlendMode:kCGBlendModeNormal alpha:A];
    }
}
```

Knihovna PaintLibrary se snaží co možná nejvíce následovat doporučení pro vykreslování 2D grafiky, a proto v metodě `drawRect:` používá minimum výpočtů s vykreslovanými objekty a minimálně tyto objekty modifikuje. V případě metody `drawRect:` v knihovně PaintLibrary dochází pouze k nastavování speciální parametrů pro renderovací kontext. Těmito parametry jsou barva pro vykreslení cesty a mód překryvu. V tomto případě je využíván mód `kCGBlendModeDefault`. Tento mód zaručuje překrytí spodních vrstev při vykreslování.

[17] Blend modes iOS https://developer.apple.com/library/mac/#documentation/graphicsimaging/conceptual/drawingwithquartz2d/dq_paths/dq_paths.html%23//apple_ref/doc/uid/TP30001066-CH211-BCIGICEF



Obrázek 13: Vykreslování objektů pomocí vrstev[18]

5.5.1.2 *Vykreslování objektů pomocí vrstev*

Jak je již naznačeno na obrázku na straně [33], při vykreslování pomocí vrstev dochází k efektům, jako je vržení stínu na objekty, které byly vykresleny dříve. Toto vyobrazení považujeme za korektní v případě, že tyto objekty nebyly vykreslovány ve stejný okamžik a měly by být tudíž považovány za několik objektů. V opačném případě by byly vnímány jako jeden objekt.

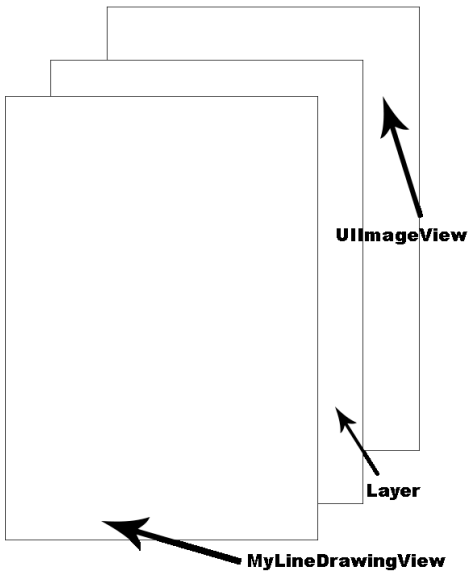
V knihovně PaintLibrary dochází také k renderování pomocí jednotlivých vrstev, jelikož toto je efektivnější a mnohem rychlejší, než vykreslování pomocí jednoho kontextu a jeho neustálé překreslování. Volání fuknce, která vynutí vykreslení kontextu, zajišťuje delegát třídy `MyLineDrawingView`.

Výpis kódu 14: MyLineDrawingViewDelegate

```
@protocol MyLineDrawingViewDelegate <NSObject>
-(void)dropperActivatedMainThread;
-(void)render;
-(void)populateDownToolbar;
@end
```

Tento delegát zajišťuje protokol, který je vyžadován třídami, které obsahují pohled `MyLineDrawingView`. Stěžejní metodou v delegátovi je metoda `render`. Tato metoda bude implementována v hierarchicky nadřazených objektech a bude vykonávat vykreslování zachycení obrazového kontextu, který bude poté z vrstev pohledu renderován do obrázku, a ten bude sloužit jako obrazový podklad. Pro lepší vizualizaci je celá hierarchie zobrazena na obrázku níže.[13]

[18] Zdroj http://developer.apple.com/library/ios/documentation/GraphicsImaging/Conceptual/drawingwithquartz2d/Art/trans_layer2.gif



Obrázek 14: Detailní vykreslení pomocí vrstvy

Viditelnost konečného obrázku reprezentovaného objektem `UIImageView` je zajištěno tak, že všechny předchozí vrstvy a pohledy jsou transparentní.

5.5.1.3 ***Popis delegátní metody render***

Metoda render, imlementována pomocí návrhového vzoru delegát, zajišťuje vykreslení obrazového kontextu pomocí metody `captureScreen`. Tento obrazový kontext je vyrenderován do vrstvy pohledu `drawScreen` typu `MyLineDrawingView`, a poté je z této vrstvy převeden do obrázku.

V této metodě je také zajištěno uvolnění vyrenderovaného obrázku a vytvoření a ukončení obrazového kontextu.

Za další funkci metody můžeme označit hlídání počtu vykreslených obrázků. Pro lepší stabilitu aplikace a nezahlcování vnitřní paměti telefonu je zajištěno, aby počet obrázků, které mohou být použity pro navrácení předchozích stavů (jedná se o funkci krok zpět), byl omezen na 5.

Tato metoda je implementována pohledem `PaintLibraryResultCompose`.

Výpis kódu 15: Paint library – metoda render

```
-(void)render{
    [self captureScreen];
    [drawScreen removeall];
}
-(void)captureScreen{
    numberOfDrawnPaths ++;
    UIGraphicsBeginImageContext(drawScreen.frame.size);
```

```
    [[drawView layer]
renderInContext:UIGraphicsGetCurrentContext()];
    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    [pathArray addObject:image];

    image = nil;
    [image release];

    drawImage.image = (UIImage*)[pathArray lastObject];
    [drawScreen setEnded:NO];

    if (pathArray.count > 6){
        [pathArray removeObjectAtIndex:0];
    }
}
```

Popisem těchto základních funkcí jsme obsáhli jádro knihovny, za které je považováno vykreslování dotyků uživatele za pomocí Cocoa Touch frameworku, a renderování takto vytvořeného grafického kontextu za pomocí vrstev do výsledného obrázku. Renderování do obrázku probíhá z důvodu potřeby následných operací s obrázkem. Těmito operacemi rozumíme ukládání, mazání, sdílení pomocí sociálních sítí a jiné.

5.5.2 Správa vlastností jádra knihovny

Správou vlastností rozumíme jakoukoliv úpravu vlastností renderování knihovny, operace prováděné s obrazovým kontextem, změna vlastností kontextu, změna vykreslovaných cest, změna vykreslovaných tvarů a podobné.

Tyto funkce jsou zajištěny pomocí dvou rozšíření knihovny, tzv. panelů nástrojů.

Horní panel nástrojů je reprezentován třídou `ToolbarController`. Tato třída dědí z třídy `UIView` a je přímo integrována do pohledu `MyLineDrawingView`. Důvodem přímé integrace a vztahu mezi těmito třídami je fakt, že tento panel nástrojů ovládá základní vlastnosti tzv. jádra.

Těmito vlastnostmi jsou změna barvy, využití gumy, změna tloušťky linky, změna alfa kanálu, změna jasu barvy a v neposlední řadě změna tvaru vykreslení uživatelské interakce a využití funkce kapátka. Další funkcí knihovny je ovládání zobrazení dolního panelu nástrojů.

Pro ovládání vlastností jádra knihovny je využito návrhového vzoru delegát, který je implementován pomocí vyžadovaného protokolu `ToolbarControllerDelegate`.

Výpis kódu 16: ToolbarControllerDelegate

```
@protocol ToolbarControllerDelegate <NSObject>
@required
-(void)changeSize:(float)laneSize;
-(void)changeAlpha:(float)alpha;
-(void)changeBrightness_delegate:(float)alpha;
-(void)gumaPress;
-(void)dropperActivated;
-(void)activateShapes:(int)shapeDeffinition;
-(void)activateDownToolbar;
@end
```

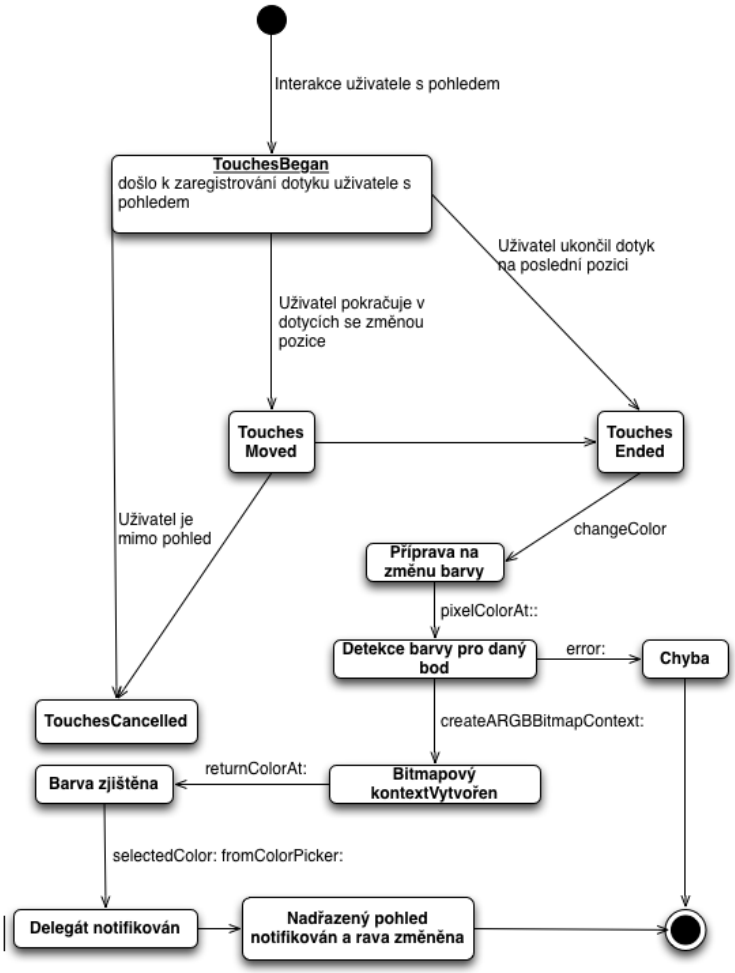
5.5.2.1 ***Změna barvy***

Jednou ze základních vlastností pr oeditaci 2D grafiky je změna barvy vykreslování. Tyto změny jsou v rámci knihovny zajištěny pomocí pohledu `CRColorPicker`.[32] Tento pohled byl vytvořen za účelem správy barev a jejich modifikací. Metody této třídy byly oproti původnímu návrhu značně modifikovány. Došlo k opravám chyb ve vytváření `CreateARGBBitmapContext` metodě.[19] Za další úpravy mohou být považovány úpravy v získávání střední hodnoty ukazatele barvy, změny výpočtů rámce pohledu a další. Tento pohled je poskytován v rámci BSD licence, ovšem je poskytován se značnými nedostatky.

Barvy jsou reprezentovány pomocí obrázku, ze kterého je poté pomocí ukazatele získávána výsledná barva. Při ukončení interakce s tímto pohledem je volána metoda `changeColor`. Tato obslužná metoda dále volá metodu `pixelColorAt::`, která přijímá argumenty typu `CGPoint` a `UIImage`. Struktura bodu reprezentuje střední hodnotu ukazatele a typ `UIImage` reprezentuje zdrojový obrázek. Tato metoda dále zajistí již zmiňované vytvoření bitmapového kontextu a dle tohoto kontextu dojde ke zjištění barvy pro konkrétní bod dotyku, tedy střední hodnotu rámce ukazatele.

Zjištěná barva je dále poskytnuta protokolu tohoto pohledu, který poskytnuté informace dále deleguje nadřazeným pohledům, které ovládájí jádro knihovny.

[19] Zdroj: http://developer.apple.com/library/mac/#qa/qa1509/_index.html



Obrázek 15: Změna barvy - stavový diagram

5.5.2.2 Aktivace funkce guma

Funkce gumy je v knihovně zpracována velmi jednoduše a efektivně. Jelikož Původní pozadí obrázku je stanoveno jako bílé, je i guma standartně stanovena jako bílá.

Efekt gumy je dosažen tak, že výchozí barva pro interakci uživatele je nastavena jako bílá. Viz změna barvy.

5.5.2.3 Změna tloušťky linky

Tloušťka linky je jedna ze základních vlastností definice kontextů a beziérových cest. Tato vlastnost je lehce editovatelná díky přístupovým metodám a atributům poskytovaným kontexty a beziérovými cestami.

Při použití kontextu můžeme tloušťku vykreslované linky nastavit pomocí funkce `CGContextSetLineWidth`. Naopak s využitím beziérových cest nastavujeme linku pomocí vlastností objektu `UIBezierPath`, a sice atributu `lineWidth`, který je definován jako datový typ `CGFloat`.

V rámci knihovny je změna tloušťky linky zajištěna pomocí posuvníku, který umožňuje změnu tloušťky linky od hodnoty 5.0 do 80.0. Zpráva o změně hodnoty posuvníku je předávána pomocí protokolu a poté dochází ke změně vnitřního stavu jádra knihovny a vlastností pohledu `MyLineDrawingView`.

Více informací o `CGFloat` lze nalézt zde [4].

5.5.2.4 ***Změna alfa kanálu***

Změna alfa kanálu je podobně jako změna tloušťky linky jeden ze základních efektů podporovaných ve standartních kreslících editorech, kterými jsem se také inspiroval. Standartně se tato funkce popisuje jako změna průhlednosti. V rámci aplikace tato funkce umožňuje postupné vrstvení a mísení jednotlivých barev, vrstev které jsou renderovány.

Jelikož je knihovna PaintLibrary komponována tak, že je užito UIColor s RGBA barevným modelem, změna alfa kanálu je provedena jako změna A komponenty RGBA modelu. Při užití tohoto modelu je dle dokumentace stanoveno, že alfa kanál je v rozsahu 0.0 – 100% průhlednost a 1.0 – 0% průhlednost.

Notifikace o změně stavu je prováděna pomocí definovaného protokolu `ToolbarControllerDelegate`.

Více informací o `UIColor` nlze nalézt zde [4].

5.5.2.5 ***Změna jasu barvy***

Jas barvy, využívá HSLA barevného modelu společně s RGBA barevným modelem. V tomto případě je nejprve nutné objekt typu UIColor převést z barevného modelu RGBA do barevného modelu HSLA a naopak.

Výpis kódu 17: Konvertování v rámci barevných modelů

```
-(void)convertFromRGB{
    UIColor *c =[UIColor colorWithRed:R green:G blue:B alpha:A];
    float h,l,s;

    [c getHue:&h saturation:&s brightness:&l alpha:&A];
    H=h;
    S=s;
    L=l;
}
-(void)ConvertFromHSL{
    UIColor *c =[UIColor colorWithHue:H saturation:S brightness:L
alpha:A];
    float r,g,b;
```

```
    [c getRed:&r green:&g blue:&b alpha:&A];
    R=r;
    G=g;
    B=b;
}
```

Po této konverzi můžeme pracovat s požadovanou komponentou barvy, a to s komponentou L (lightness). Tato komponenta umožňuje změnu jasu barvy. Poté však musí být aktualizována užívaná barva. Tuto funkčnost opět zajišťuje již zmiňovaný protokol.

5.5.2.6 ***Kapátko***

Kapátko je jednou z pamětově a exekučně náročnějších funkcí. Standartně tato funkce zajišťuje tzv. „odkapání“ barvy z kontextu. Cílem bylo zajistit, co možná nejefektivnější splnění požadavku.

Dle dokumentace byly implementovány metody pro získání ARGBBitmapového kontextu a také pro získání barvy daného pixelu. Tyto metody byly implementovány dle dokumentace společnosti Apple. Problémem však je, že knihovna nemohla být testována se stylusem a také to, že většina běžných uživatelů stylus pro zařízení nevlastní. Tudíž vyvstává problém, že místo dotyku často neodpovídá požadovanému místu dotyku. Tento problém lze řešit již zmíněným stylusem .

Jelikož zařízení, na kterých byla knihovna testována, využívají nativně ARGB barevný model, je možno z již vytvořeného kontextu vyjmout komponenty na určitých místech. Tyto komponenty jsou řazeny v pořadí ARGB.

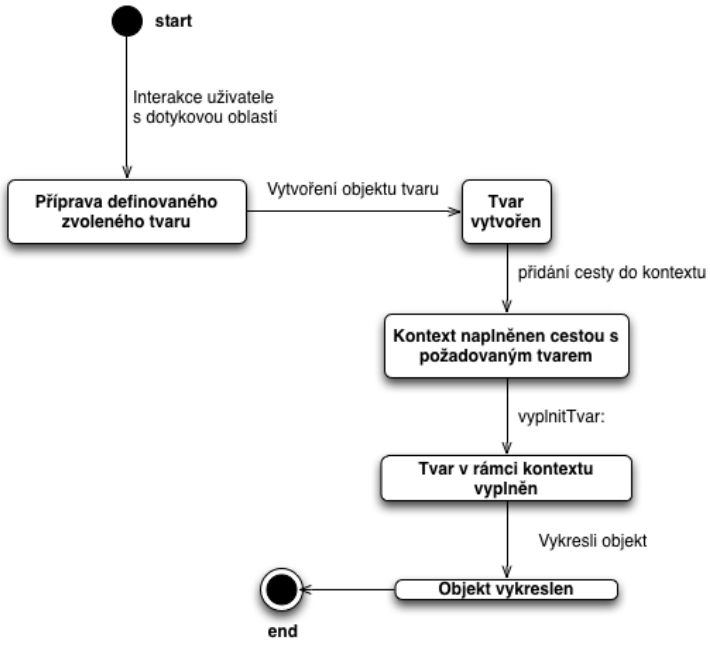
Získání těchto komponent dosáhneme tak, že vytvoříme ARGBKontext, dále zjistíme jeho výšku a šířku a dle dokumentace vypočteme odsazení pro daný bod dotyku. Tímto dostaneme první komponentu – Alfa kanál. Na místě odsazení +1 nalezneme komponentu R – tedy červenou barvu a na místech odsazení + 2 a odsazení +3 dále G a B komponenty.

Z těchto komponent můžeme tedy složit požadovanou barvu a nastavit ji jako výchozí pro další interakci uživatele s plátnem.

5.5.2.7 ***Definované tvary***

Knihovna sama o sobě obsahuje již předem definované tvary. Těmito tvary jsou obdelník, trojúhelník, ovál, kruh a v neposlední řadě hvězda. Všechny tyto tvary jsou zděděny ze třídy `PaintLibraryShape`. Jejich společnou metodou je metoda `draw:::`, která přijímá argumenty `CGRect`, `CGFloat (float)` a `UIColor`. Následně v této metodě probíhá vytvoření požadovaného tvaru pomocí `CGContext`, `UIBezierPath`. `ToolbarControllerDelegate`, který byl popsán již dříve, poskytuje reakční a komunikační metody pro změnu tvaru vykreslovaného objektu. Výchozím tvarem je linka. Výchozí tvary jsou vyplňovány pomocí funkce `CGContextFillPath`.

Definované tvary jsou navrženy tak, že při interakci uživatele dojde k vykreslení pouze jednoho tvaru pro každý dotyk uživatele.



Obrázek 16: Vykreslení předdefinovaného tvaru

5.5.3 Rozšiřující vlastnosti knihovny

Za tyto vlastnosti můžeme označit vyčištění plátna, export obrázku do formátu PNG nebo JPEG, načtení 2D grafiky z knihovny obrázků nebo fotoaparátu, sdílení vytvořené grafiky za pomocí sociálních sítí, funkci krok zpět a také vyplnění definované oblasti, tzv. vylévání barvou.

Všechny tyto vlastnosti jsou zajištěny za pomocí interakce druhého panelu nástrojů. Tento panel je v UI umístěn na spodním okraji využívaného pohledu.

5.5.3.1 DownToolbarDelegate

`DownToolbarDelegate` definuje protokol, který se stará o přeposílání požadavků pohledu, který implementuje `DownToolbar`. Tento toolbar ovládá a deleguje funkce, které byly popsány v předcházejícím odstavci 5.5.3.

Výpis kódu 18: DownToolbarDelegate

```
@protocol DownToolbarDelegate <NSObject>
@required
-(void)clearAll;
-(void)imagePicked:(UIImage*)pickedImage;
-(void)shareActionRequested;
-(void)requestPoppingView;
-(void)requestSave;
-(void)activateBucket;
```

```
-(void)requestUndo;
@end
```

Metody, uvedené a deklarované v tomto protokolu jsou vyžadovány, a tudíž musí být nevyhnutelně implementovány v pohledech, které tento protokol využívají.

Krom jiného je možno ještě metody označit klíčovým slovem `@optional`. Pokud metody označíme takto, je možno zvolit si, zda budeme metody protokolu implementovat v rámci pohledu. Je však potom nutné vždy ověřit, zda je tato metoda implementována pomocí funkce `respondsToSelector:`.

5.5.3.2 Export obrázku do formátu PNG/JPEG

Export obrázku do těchto formátu probíhá za pomocí níže uvedených funkcí. Tyto funkce převedou zdrojový obrázek typu `UIImage` na typ `NSData`. Pokud chceme využít JPEG reprezentaci, měli bychom vhodně volit kompresi obrázku. Pokud bude komprese příliš vysoká, kvalita obrázku se bude rapidně snižovat.

Při PNG obrazové reprezentaci kompresi neurčujeme. Zde máme možnost využít transparentnost.

Následně po vytvoření obrázku jej můžeme zapsat do knihovny obrázků. Tento zápis probíhá pomocí funkce `UIImageWriteToSavedPhotosAlbum()`. Tato funkce přijímá argumenty `UIImage, completionTarget, completionSelector, void`. První atribut je obrázek, jenž má být uložen, druhý parametr určuje objekt, který bude implementovat reakční metodu – atribut tři, poslední atribut zaštiťuje informace o uložení.

Výpis kódu 19: Export obrázku do JPEG / PNG

```
-(UIImage*)JPEGRepresentation:(UIImage*)img_
withTheCompressionQuality: (CGFloat) compression_quality
{
    NSData * data = UIImageJPEGRepresentation(img_,
compression_quality);
    return [UIImage imageWithData:data];
}
-(UIImage*)PNGrepresentation : (UIImage*)img_
{
    NSData * data = = UIImagePNGRepresentation(img_);
    return [UIImage imageWithData:data];
}
```

5.5.3.3 ***Načtení 2D grafiky z knihovny obrázků***

Načítání 2D grafiky je nedílnou součástí knihovny. Tato funkce je velmi užitečná pro úpravu již pořízených fotografií. Zvláštností v rámci implementace je fakt, že metody pro vyvolání výběru fotografií se liší pro zařízení iPhone i iPad.

Další zvláštností jsou různé zdroje pro import 2D grafiky. Těmito zdroji mohou být následující:

- UIImagePickerControllerSourceTypePhotoLibrary
- UIImagePickerControllerSourceTypeSavedPhotosAlbum
- UIImagePickerControllerSourceTypeCamera

Pro výběr obrázků z knihovny se používá `UIImagePickerController`. Tomuto objektu je následně přiřazen zdrojový typ z výše uvedeného výčtu zdrojů.

Výpis kódu 20: Výběr 2D grafiky z knihovny obrázků

```
-(void)pickAnImage{
     [self presentModalViewController:picker animated:YES];
}
-(void)pickIMAGEforIPAD:(UIView *) poppingView{
     loginPop = [[UIPopoverController alloc]
initWithContentViewController:picker];
     [loginPop presentPopoverFromRect:CGRectMake(self.view.frame.s
ize.width / 2 - 160, self.view.frame.size.height / 2 - 160,
320, 320) inView:poppingView
permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];
}
```

5.5.3.4 ***Metoda krok zpět – undo***

Speciální funkcí knihovny je navrácení předchozích vyrenderovaných stavů. Předchozí vyrenderované stavy obrazového kontextu jsou uchovávány jako obrázky v poli. Konkrétně tedy v poli typu `NSMutableArray`. Typ `NSMutableArray` je typ pole, které je přímo určeno pro provádění častých změn. I při častých změnách vlastností pole, je zaručeno, že vložení i vyjmutí prvku z pole proběhne v lineárním čase. Pole je navíc dynamicky alokováno.

Všechen kód pro zajištění navrácení stavu je uveden v metodě `undo`. V této metodě probíhá nejprve kontrola, zda je v obrazovém pohledu přiřazen obrázek. Pokud není, znamená to, že je buď počet obrazových kontextů roven nule nebo se jedná o prvotní zavolání metody `undo`, s tím, že doposud nebyl žádný obrazový kontext vytvořen.

Další kontrolou v rámci funkce je kontrola počtu obrazových kontextů. Když tato kontrola projde, je zaručeno, že obrazový kontext v podobě obrázku existuje a může být přiřazen obrazovému pohledu. Dojde k dekrementaci pole a přiřazení obrázku.

Výpis kódu 21: Metoda undo

```
-(void)undo{
    if (drawImage.image == nil) {
        return;
    }
    else if(pathArray.count > 0 && numberOfDrawnPaths < 6){
        [pathArray removeLastObject];
        [drawImage setImage:(UIImage*)[pathArray lastObject]];
    }
}
```

5.5.3.5 Sdílení vytvořené 2D grafiky

Knihovna nabízí možnost sdílení vytvořené grafiky. Tato vlastnost je dosažena pomocí open source projektu `ShareKit`.[33] Tato knihovna je licencována v rámci `MIT Open Source license`[20]. Detailní popis importu této knihovny do projektu je popsán na oficiálních stránkách. Knihovna zahrnuje sdílení v rámci běžných sociálních sítí jako jsou Facebook, Twitter, Tumblr, Pinboard a další. Umožňuje také odesílání 2D grafiky pomocí emailu. Knihovna je kontinuálně vylepšována a jsou jí přidávány nové a nové vlastnosti. Tato knihovna je jednou z nejpopulárnějších v ohledu sdílení informací.

Dokumentace a podpora k této knihovně je také rozsáhlá díky fóru a FAQ. Tato knihovna je podporována od verze systému iOS 4.3.x.

S uvedením systému iOS 6.x se však tato knihovna stává méně a méně využívanou, jelikož společnost Apple s touto verzí systému vydala nové frameworky. Těmito frameworky jsou Social a Twitter a tyto umožňují snadnější implementaci sociálních sítí. Nicméně, knihovna `PaintLibrary` je designována a podporována od verze systému iOS 5.x a proto stále využívá `ShareKit`.

`ShareKit` umožňuje sdílení nejen 2D grafiky. Hovoříme – li však o 2D grafice, podporovanými formáty jsou PNG a JPEG.

Zakomponování ShareKitu do Code X projektu je vcelku pracné a časově náročné. Krom jiného je nutno vytvořit na sociálních sítích aplikace, přiřadit jim správná práva a jiné vlastnosti. Společně s tímto je nutné vytvořit si vlastní konfigurační třídu v rámci projektu, která bude spravovat vlastnosti ShareKitu. V našem případě se jedná o třídu `MySHKConfigurator`. Tato třída obsahuje obslužné metody pro konfiguraci `ID` a jiných potřebných vlastností pro správu sdílení.

V následujícím výpisu kódu je znázorněno sdílení 2D grafického objektu. Konkrétně se jedná o objekt typu `UIImage`. Třída SHKItem je vnitřní třídou knihovny, která reprezentuje objekt sdílení. Tímto objektem může být obrázek, text a jiné. Následně po vytvoření objektu je vytvořen akční list,

[20] ShareKit licence http://getsharekit.com/license

který nabídne možnosti sdílení pro daný objekt. V našem případě se bude nejčastěji jednat o služby, které jsou schopny přijmout obrázek.

Výpis kódu 22: Ukázka sdílení obrázku

```
-(void)share:(UIImage*)image :(UIView*)view :(CGRect)frame{

SHKItem *item = [SHKItem image:image title:SHKLocalizedString(@"Lo
ok at this picture painted with PaintLibrary app!")];
        // Get the ShareKit action sheet
SHKActionSheet *actionSheet = [SHKActionSheet actionSheetForItem:i
tem];
        // Display the action sheet
    [actionSheet showFromRect:frame inView:view animated:YES];
}
```

5.5.3.6 Vylévání barvou

Vylévání barvou ang. „paint bucket“ je jednou ze základních funkcí 2D grafických editorů. Tato funkce je hodně využívána ve standartních grafických editorech. Problémem v rámci mobilních zařízení je však paměťová náročnost.

Nejčastěji využívanými algoritmy pro implementaci této funkce jsou následující:

- FloodFill
- SeedFill[34]
- ScanLine[35]
- ScanLine v kombinaci s FloodFill

Tyto algoritmy jsou velmi jednoduché na implementaci, ovšem ve většině případů jsou efektivní díky využití rekurze. V případě implementace těchto algoritmů v rámci mobilních zařízení není vhodné používat rekurzi, jelikož ta je paměťově velmi náročná.

Doporučuje se také využívat tuto funkci pro vyplňování menších ploch, jelikož pro velké plochy je vcelku neefektivní. V knihovně je tento problém vyřešen tak, že je vytvořena instance `UIBezierPath` a tato cesta je postupně plněna v místech, kde se barva pro daný pixel liší. Tato cesta je po dokončení vyplněna požadovanou barvou. `PaintLibrary` využívá algoritmu FloodFill, jelikož je počítáno s vyplňováním rozumně velkých oblastí. Pro větší oblasti by byla vhodnější implementace algoritmu ScanLine společně v kombinaci s FloodFill.

O vyplnění oblasti se stará třída `FloodFillContext`. Konkrétně tedy metoda `floodFill`. Tato funkce využívá implementace čtyř směrového algoritmu `FloodFill`.

Výpis kódu 23: FloodFill implementace

```
-(void)floodFill:(CGPoint) mypoint
{
    BOOL isOutOfBounds = [self isOutOfBounds:mypoint];
```

```
    if ( isOutOfBounds) {
        return;
    }
    BOOL colorEquality = [self compareColors:[self
getPixelINTColorAtLocation:mypoint]];
    if (colorEquality == NO) {
        return;
    }
    if ([path containsPoint:mypoint])
        return;
    }
        [path moveToPoint:mypoint];
        [path addLineToPoint:mypoint];
        [self floodFill:CGPointMake(mypoint.x+lane/2, mypoint.y)];
        [self floodFill:CGPointMake(mypoint.x-lane/2, mypoint.y)];
        [self floodFill:CGPointMake(mypoint.x, mypoint.y+lane/2)];
        [self floodFill:CGPointMake(mypoint.x, mypoint.y-lane/2)];
}
```

V rámci řešení funkce vylévání barvou bylo provedeno mnoho optimalizací. Tyto optimalizace zahrnují čištění paměti, optimalizaci výsledného kódu, optimalizaci procházení polem a v neposlední řadě zachytávání úniků paměti. Tyto metody budou podrobněji popsány v následující kapitole.

5.6 Optimalizace použitých technologií

Ve fázi implementace byla knihovna průběžně testována. Ne vždy byly výsledky testování uspokojující. Dlouhým a náročným problémem bylo vykreslování jemných a kontinuálních linek. Dle dokumentace se doporučuju využití `CGContextRef`, ovšem dle zjištění v rámci testování je tato struktura nevyhovující. Množství alokací se využitím `CGContextRef` rapidně zvyšuje oproti využití výsledných `UIBezierPath`. Důvodem je neustálá alokace obrazového kontextu.

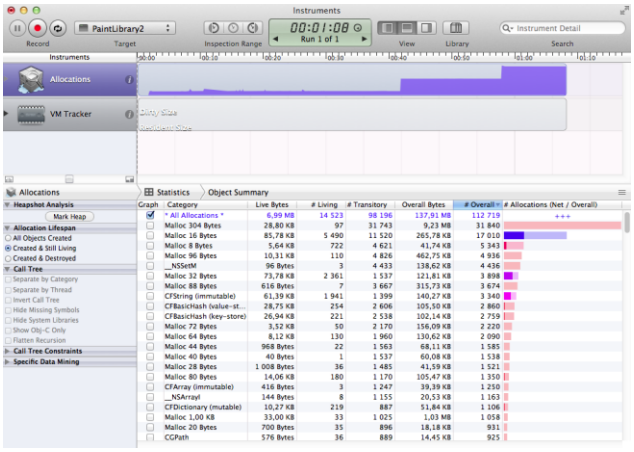
Velmi spolehlivou a užitečnou aplikací pro testování a optimalizace je aplikace Instruments.[21] Tato aplikace umožňuje sledování alokací, úniků paměti, takzvaných „zombie[22]“ objektů[23], sledování

[21] Instruments http://developer.apple.com/library/mac/#documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html

[22] NSZombie http://cocoadev.com/wiki/NSZombie

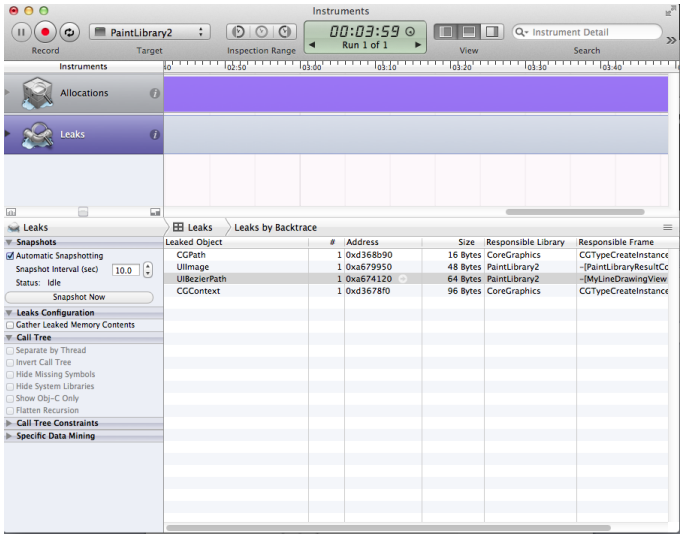
[23] Zombie objekt je objekt, který v paměti zůstane i přes jeho korektní uvolnění

aktivity CPU a také množství využití energie. Díky přímé integraci v Xcode IDE je možné přímo vysledovat části kódu, které způsobují problémy.



Obrázek 17: Náhled aplikace Instruments

Při testování úniků paměti se většina úniků většinou vyskytuje při alokacích. Někdy jsou tyto úniky dokonce způsobeny i samotným kódem v rámci poskytovaných knihoven společnosti Apple.



Obrázek 18: Testování úniků paměti

Knihovna PaintLibrary prošla několika testy a byla vyvinuta snaha o odstranění co možná největšího množství zjištěných problémů. Mezi nejčastější problémy patřily:

- Nadměrná alokace paměti
- Úniky paměti při vytváření objektů
- Dealokace objektů v nesprávnou chvíli
- Nadměrné využití paměti zařízení

K odstranění těchto problémů byla využita právě aplikace Instruments. Další velmi výraznou pomocí je využití tzv. globální zarážky. Tato zarážka v Xcode IDE umožňuje zobrazení stack trace při jakékoliv chybě. Tato funkce velmi usnadňuje práce s Xcode IDE.[24]

Provedené optimalizace se týkaly především vykreslování a vykreslovacího kódu. Dle dokumentace je uvedeno, že veškerý kód by měl být umístěn v metodě `drawRect:`, ovšem toto není tak efektivní, jako umístit dílčí kusy kódu do pomocných metod nebo si dokonce předpřipravit obrazový kontext v pomocných metodách. Umístění kompletního vykreslovacího kódu, který může ve výsledku obsahovat velké množství funkcí a nastavování parametrů, vede ke značnému zpomalení aplikace. Proto je v rámci PaintLibrary v metodě `drawRect:` umístěn pouze nejnutnější kód pro samotné vykreslení objektů.

Další optimalizací provedenou ve vykreslovací metodě je nahrazení `CGContextRef`. `UIBezierPath` poskytuje jemnější vykreslování linek a je také stabilnější pro následné použití a znovu využití.

Následné optimalizace se týkaly efektivnosti kódu a alokací objektů. Nejvíce optimalizovanou části bylo jádro knihovny. Pro optimalizaci úniků paměti při alokacích bylo využito `dispatcheru`.

Ačkoliv dokumentace doporučuje využití určitých metod a způsobů vykreslování, při testování bylo zjištěno, že existují efektivnější postupy pro vykreslování 2D grafiky.

[24] Exception breakpoint http://developer.apple.com/library/mac/#recipes/xcode_help-breakpoint_navigator/articles/adding_an_exception_breakpoint.html

6 Závěr

Cílem mé bakalářské práce bylo zjistit trendy na poli vývoje pro mobilní platformu iOS v rámci editace 2D grafiky, porovnat již existující řešení a navrhnout vlastní řešení s možností použití některého z již existujících řešení. Průběh návrhu takovéhoto řešení, jeho implementace a testování je popsán v předešlých kapitolách. V této kapitole bych se rád zmínil, v jakém stavu je současné řešení a jaké jsou další možnosti rozšíření vytvořené knihovny.

6.1 Aktuální stav

V této chvíli je knihovna připravena k použití. Cíle, jenž byly stanoveny, byly naplněny. Knihovna splňuje základní požadavky stanovené v rámci návrhu. Obsahuje algoritmy pro vykreslování 2D grafiky, její editaci a uchovávání. Obsahuje také funkci vylévání barvou a vykreslování základních tvarů.

Knihovna je sestavena tak, aby pro programátora bylo co možná nejjednodušší ji použít a případně dále rozšiřovat.

Knihovna byla zahrnuta v ukázkové aplikaci, která byla vytvořena pro demonstraci vlastností knihovny.

6.2 Možnosti dalšího vývoje

V budoucnu plánuji rozšíření možností pro editaci 2D grafiky. Plánuji také přidání některých standartních nástrojů, jako je například výběr oblasti, ořezávání a podobné. Další možností pro doplnění knihovny je vytvoření pohledu, který by byl schopen ovládat vícenásobné dotyky uživatele a jejich vykreslování. Zajímavým rozšířením by se mohla stát komponenta, která by umožňovala vytváření vlastních 2D předdefinovaných objektů pomocí speciálních nástrojů.

Rád bych také prozkoumal možnosti editace 3D grafiky a případně tuto knihovnu rozšířil o možnosti editace 3D objektů a grafiky.

Rád bych prozkoumal možnosti napojení této knihovny na webové API a to tak, že by bylo umožněno více uživatelům v současnou chvíli editovat, či vytvářet 2D grafiku.

Použitá literatura

[1] KOCHAN , Stephen G. Programming in Objective-C 2.0 2nd ed. Upper Saddle River: Addison Wesley Professional, c2009, xv, 600 s. ISBN 978-0-321-56615-7

[2] MARK, Dave. Begginning iPhone development: exploring the iPhone SDK. Berkely: Apress, c2009, xxiii, 508 s. ISBN 1430216263

[3] Ray Wenderlich. RAYWENDERLICH [online]. 2012 [cit. 2013-04-29]. Dostupné z: http://www.raywenderlich.com

[4] APPLE, Inc. Apple Developer iPhone OS Reference Library [online]. 2010 [cit. 2013-04-29]. Dostupné z: http://developer.apple.com/iphone/library/

[5] Mac Developer Library: Transitioning to ARC Release Notes? Mac Developer Library [online]. [2013-04-29]. Dostupné z: http://developer.apple.com/library/mac/#releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html

[6] CGPath Reference. APPLE, Inc. Mac Developer Library [online]. 2010 [cit. 2013-04-29]. Dostupné z: https://developer.apple.com/library/mac/#documentation/graphicsimaging/Reference/CGPath/Reference/reference.html

[7] APPLE INC. Apple Developer: Quartz 2D Programming Guide [online]. 2010 [cit. 2013-04-29]. Dostupné z: https://developer.apple.com/library/mac/#documentation/GraphicsImaging/Conceptual/drawingwithquartz2d/Introduction/Introduction.html

[8] GitHub: Animated Paths [online]. 2013 [cit. 2013-04-29]. Dostupné z: https://github.com/ole/Animated-Paths

[9] Code4App: Paint Pad [online]. 2012 [cit. 2013-04-29]. Dostupné z: http://code4app.net/ios/Paint-Pad/4fcf74876803faec66000000

[10] APPLE INC. Apple Developer: OpenGL ES Programming Guide [online]. 2010 [cit. 2013-04-29]. Dostupné z: http://developer.apple.com/library/ios/#documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/Introduction/Introduction.html

[11] GitHub: SmoothLineView [online]. 2013 [cit. 2013-04-29]. Dostupné z: https://github.com/levinunnink/Smooth-Line-View

[12] GitHub: Cloud Finger Paint [online]. 2013 [cit. 2013-04-29]. Dostupné z: https://github.com/Atrac613/CloudFingerPaint-Client-iOS

[13] Youtube: Cloud Finger Paint iOS demo [online]. 2011 [cit. 2013-04-29]. Dostupné z: http://www.youtube.com/watch?v=zt9FDD4NCaw

[14] GitHub: CeedGL [online]. 2013 [cit. 2013-04-29]. Dostupné z: https://github.com/rsebbe/CeedGL

[15] GitHub: SmoothDrawing [online]. 2013 [cit. 2013-04-29]. Dostupné z: https://github.com/krzysztofzablocki/smooth-drawing

[16] Sparrow: Forum [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://forum.sparrow-framework.org

[17] Sparrow: Help [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://gamua.com/sparrow/help/

[18] Sparrow Wiki: SHLine Extension [online]. 2011 [cit. 2013-04-29]. Dostupné z: http://gamua.com/sparrow/help/

[19] Cocos2D [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://www.cocos2d-iphone.org

[20] Cocos2D: DrawLines [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://gamedev.sugartin.info/2011/12/16/45/

[21] APPLE INC. IOS [online]. 2013 [cit. 2013-04-30]. Dostupné z: http://www.apple.com/ios/what-is/

[22] About.com [online]. 2013 [cit. 2013-04-30]. Dostupné z: http://ipod.about.com/od/iphonesoftwareterms/qt/apps-in-app-store.htm

[23] iTunes: Temple Run [online]. 2013 [cit. 2013-04-30]. Dostupné z: https://itunes.apple.com/us/app/temple-run/id420009108?mt=8

[24] ColorPicker [online]. 2013 [cit. 2013-04-30]. Dostupné z: http://www.colorpicker.com

[25] WhatIs.com: MVC [online]. 2013 [cit. 2013-04-30]. Dostupné z: http://whatis.techtarget.com/definition/model-view-controller-MVC

[26] APPLE INC. Developer Apple: Cocoa Design Patterns [online]. 2010 [cit. 2013-04-30]. Dostupné z: https://developer.apple.com/library/mac/#documentation/cocoa/conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html

[27] Observer Pattern [online]. 2013 [cit. 2013-04-30]. Dostupné z: http://www.oodesign.com/observer-pattern.html

[28] Programming for iOS: Dessign Patterns - KVO [online]. 2013 [cit. 2013-04-30]. Dostupné z: http://www.oodesign.com/observer-pattern.html

[29] APPLE INC. Cocoa Fundamentals: Design Patterns [online]. 2010 [cit. 2013-04-30]. Dostupné z: https://developer.apple.com/library/mac/documentation/cocoa/conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html#//apple_ref/doc/uid/TP40002974-CH6-SW27

[30] APPLE INC. Dispatch Queues [online]. 2012 [cit. 2013-04-30]. Dostupné z: http://developer.apple.com/library/ios/#documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html

[31] APPLE INC. Developer Apple: UIBezierPath Class Reference [online]. 2012 [cit. 2013-04-30]. Dostupné z: http://developer.apple.com/library/ios/#documentation/uikit/reference/UIBezierPath_class/Reference/Reference.html

[32] Color Picker [online]. 2011 [cit. 2013-04-30]. Dostupné z: http://icrglabs.wordpress.com/2011/09/20/color-picker-for-iphoneipad/

[33] ShareKit [online]. 2013 [cit. 2013-04-30]. Dostupné z: http://getsharekit.com

[34] Scan Line Seed Fill Algorithm. OnlineMCA.com [online]. 2009 [cit. 2013-04-30]. Dostupné z: http://onlinemca.com/mca_course/kurukshetra_university/semester5/computergraphics/scan_line_seed_fill.php

[35] Scanline Fill Algorithms. [online]. [cit. 2013-04-30]. Dostupné z: http://www.cecs.csulb.edu/~pnguyen/cecs449/lectures/fillalgorithm.pdf

Seznam příloh

Součástí BP je CD.

/sources	Ze jsou uloženy zdrojové kódy knihovny
/sources/library	Zde jsou uloženy zdrojové kódy knihovny PaintLibrary
/sources/demo_app	Zde je k nalezení ukázková aplikace
/thesis	V tomto adresáři je přiložen tento text

Výpis kódu 24: CeedGL vykreslování - ukázka

```
NSString *vertexShaderSource =
STRINGIFY(
          attribute vec4 a_position;
     attribute vec4 a_source_color;
          varying vec4 v_destination_color;
          uniform mat4 u_projection_matrix;
          uniform mat4 u_modelview_matrix;

          void main(void)
          {
              v_destination_color = a_source_color;
   gl_Position = u_projection_matrix * u_modelview_matrix *
a_position;
          }
          );
NSString *fragmentShaderSource =
STRINGIFY(
          varying lowp vec4 v_destination_color;
          void main(void)
          {
              gl_FragColor = v_destination_color;
          }
          );
program = [[GLProgram program] retain];
NSError *error = nil;
GLShader *vshader = [GLShader vertexShader], *fshader = [GLShader
fragmentShader];

[vshader setSource:vertexShaderSource];
if(![vshader compile:&error]) { NSLog(@"Vertex shader compilation
error: %@", error); }

[fshader setSource:fragmentShaderSource];
if(![fshader compile:&error]) { NSLog(@"Fragment shader
compilation error: %@", error); }
```

```
[program attachShader:vshader];
[program attachShader:fshader];
[program bindAttributeLocation:0 forName:@"a_position"];

if(![program link:&error])  {
NSLog(@"Could not link program error: %@", error);
}
command = [[GLDrawCommand drawCommand] retain];
command.program = program;
command.mode = GL_TRIANGLES;
command.firstElement = 0;
command.elementCount = 3;
GLBuffer *attr = [GLBuffer buffer];
GLfloat pos[] = {0,0,   0.5,0,  0.5,1};
[attr loadData:pos size:sizeof(pos) usage:GL_STATIC_DRAW
target:GL_ARRAY_BUFFER];
[command setAttributeBuffer:attr size:2 type:GL_FLOAT
normalized:GL_FALSE stride:0 offset:0 forName:@"a_position"];

[command setAttribute:[GLValue vectorWithFloats:1 :0 :0 :1]
forName:@"a_source_color"];
GLfloat modelView[] = {1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1};
GLfloat projection[] = {1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1};
[command setUniform:[GLValue matrixWithFloats:modelView size:4]
forName:@"u_modelview_matrix"];
[command setUniform:[GLValue matrixWithFloats:projection size:4]
forName:@"u_projection_matrix"];
- (void)drawFrame
{
    [(EAGLView *)self.view setFramebuffer];
    glClearColor(0, 0, 0.3, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    [command draw];
}
```

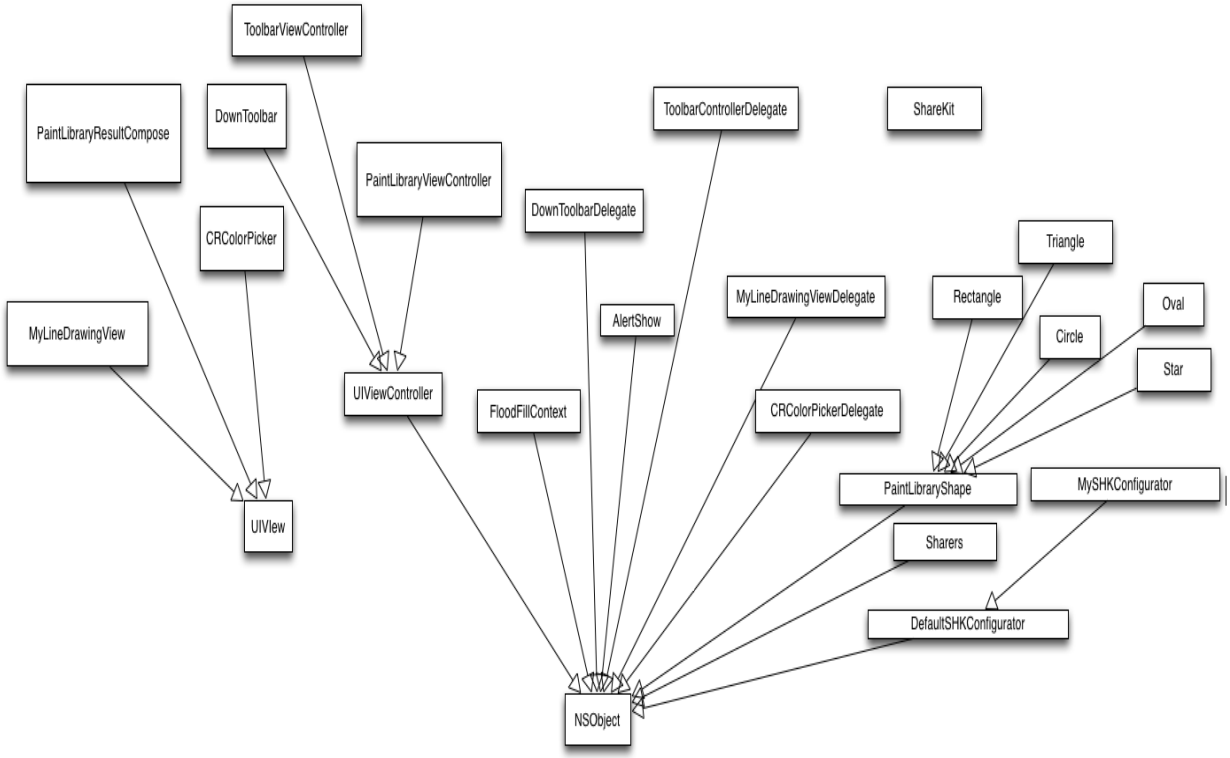
Příloha.B: Zařízení použitá pro testování

Tabulka.2: Zařízení použitá pro testování

	iPad 1	**iPhone 4S**	**MacBook Air**
Model	1st generation	4th generation	13“
OS verze	iOS 5.1.1	iOS 6.1.3	OS X 10.8.3
Paměť	16 GB	16 GB	128 GB SSD

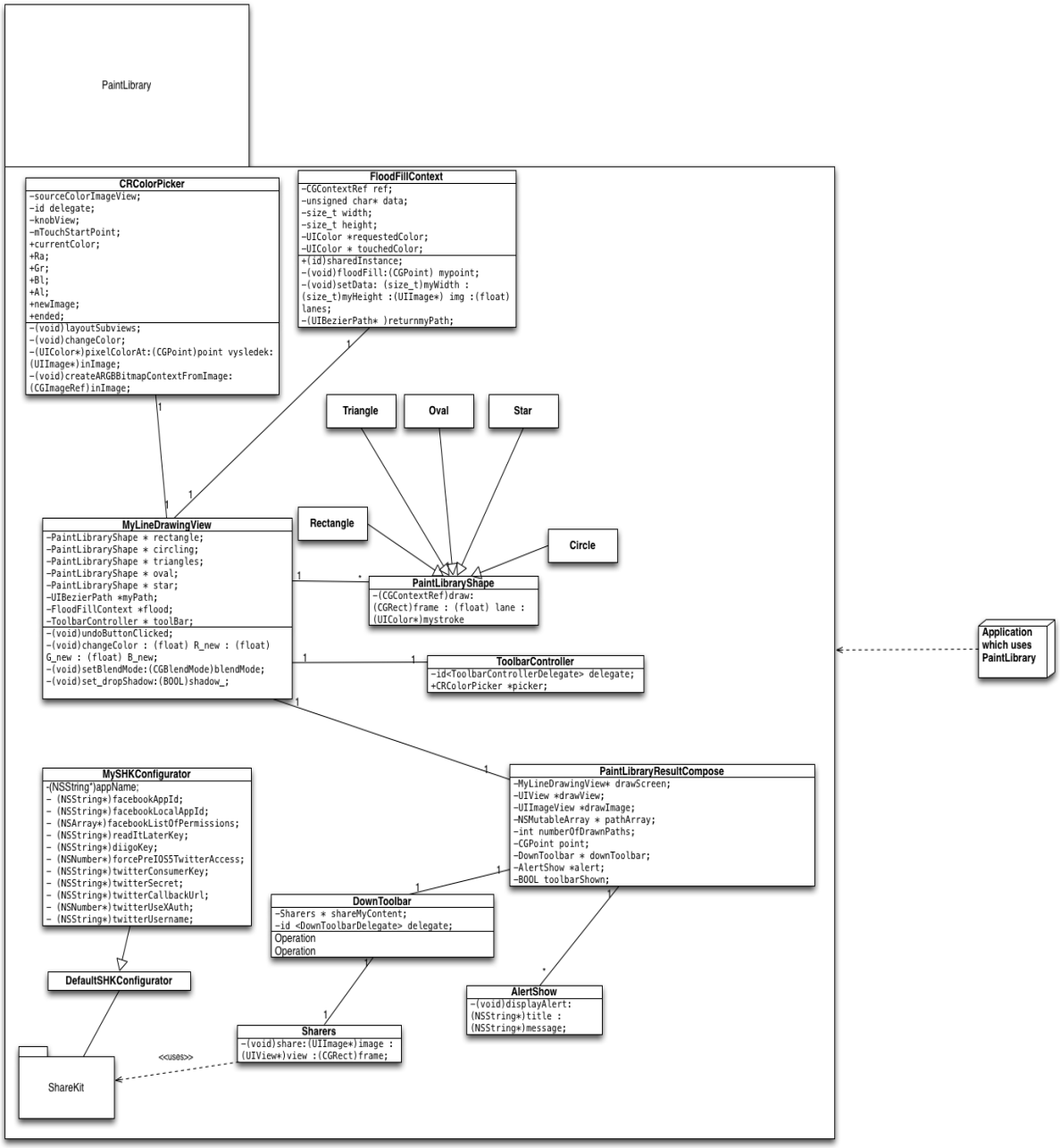
Tato zařízení byla využita se souhlasem jejich majitelů. Testovací zařízení MacBook Air není považováno za relevantní z důvodu neodpovídajícího výkonu reálným zařízením. Z tohoto ohledu nebylo toto zařízení zohledňováno v rámci prokazatelných testů.

Příloha.C: Třídní diagram Paint Library



Obrázek 19: Třídní diagram PaintLibrary

Příloha.D: PaintLibrary UML diagram



Obrázek 20: PaintLibrary UML